```
************************************************************
                                  ***
        /////////      ////   ////  ***  /////////
       IIIIIIIII/     IIII/  IIII/    ***IIIIIIII//
      IIIII/          IIII/  IIII/     IIII/  IIII/
      IIIII/          IIII/  IIII/     IIII/* IIII/
      IIIII/          IIII/  IIII/     IIII/***III/
      IIIII//////  // IIII///IIII/  // IIII///***I/  //
       IIIIIIIII/ II/  IIIIIIIII/  II/  IIIIIIII*** II/
                                                 ***
************************************************************
                 CIRCULO DE USUARIOS DE QL
************************************************************
```

Fanzine mensual independiente para usuarios de Sinclair QL y compatibles

AÑO 3 — NUMERO 17 — FEBRERO DE 1990

Estamos en el año 10 Después de Sinclair. Toda Hispania está ocupada por las legiones de PCs. ¿Toda? ¡No! Un puñado de irreductibles QLs resiste todavía y siempre ante el primitivo invasor...

Compilación de colaboraciones y distribución: Salvador Merino
Para recibir información sobre cómo recibir y/o colaborar en el fanzine, enviad un sobre franquedo y con vuestra dirección a: Marcos Cruz, Acacias 44, 28023 MADRID.

CONTENIDO

Portada de este número: 3D PRECISION

Con este número de CUQ se incluyen los programas siguientes:

- OVERDRIVE_BAS (autor: Sergio Montoro Ten)
- Toolkit LISP (autor: " " ")
- Edisc101_exe.- Nueva versión (Autor: José Carlos de Prada)

Material preparado para próximos números:

- La sección ASM vuelve con un viejo tema relacionado con volcados de pantallas a impresora con grises (versiones 8 agujas y 24 agujas).
- Manual PCB2
- Empieza el proyecto FORTH 32 bits NO STANDARD (MERINO-FORTH).
- Comentario PC CONQUEROR

EDITORIAL

Según se lee por ahí, 1989 ha sido un año muy malo para casi todo el mundo. Menos para nosotros que todavía seguimos vivos, según se mire.

Por lo que se vé ya ha salido al mercado los primeros PCs basados en el nuevo INTEL 80486, el cual visto sobre el papel parece muy potente, pero corriendo MS-DOS, la verdad no hace falta gastarse 2 o 3 millones de ptas.

Disponible OS/2 EE 1.1, el cual requiere como minimo un 286 y 3 ó 4 Megas de Memoria (+ un disco duro de alta capacidad). Permite ejecutar 17 tareas distintas (sólo 12 en la versión standard 1.0). En una máquina con 3 Megas el OS/2 deja libre después de cargarlo un raquitico Mega teniendo que recurrrir a un gestor de memoria virtual en el disco duro. El OS/2 es difícil de instalar. A veces exasperante. Configurarlo correctamente puede colmar la paciencia de cualquiera, y sacarle un mínimo de partido requiere estudiar un muy considerable volumen de documentación bastante técnica.

Todo lo anterior ha sido sacado de EL ORDENADOR PERSONAL Diciembre/Enero 1989/90. Pero en un articulo escrito por un usuario de PC, titulado Ordenadores domésticos : pioneros y parias, dice que el QL no podia desempeñar labores serias de gestión. Podia llevar un registro de los libros o discos que tenemos en casa, pero de ningún modo era posible utilizarlo en una oficina o pequeña empresa (salvo que ésta fuera realmente pequeña). Por tanto se quedó a medio camino entre el ocio y la gestión y no tuvo el éxito esperado.

A mi manera de pensar, ese viejo usuario de ZX, ahora PC, nunca se ha enterado realmente de que ocurrió con el QL en Marzo de 1986 y cuál ha sido su suerte hasta la presente. Programas como el PC CONQUEROR ponen en ridiculo al IBM PC y los emuladores de QL en el Atari ST y AMIGA, compatibles QLs como el THOR y el ATARI SMS2 ponen en evidencia que el QL y su sistema operativo QDOS multitarea no ha sido olvidado por un gran número de usuarios muy contentos a pesar de todo.

S. Merino

NOTICIAS

EL EMULADOR DE QL EN EL COMMODORE AMIGA

El pasado mes de diciembre un usuario de QL 128K sin ampliar y Amiga, estaba interesado en escribir un emulador de QL en el Amiga. Le dije que era posible modificando las rutinas del teclado e interface serie del listado en assembler comentado de la ROM del QL, la pantalla se podria solucionar con un job que la adaptará a formato Amiga cada cierto tiempo, pero donde habria alguna dificultad seria en escribir las rutinas para controlar el disco del Amiga en formato QDOS ya que las rutinas que poseo son referente a MDVs. Y Juan José Ramirez Lozano (C/ Esperanza,8 , EDF "Sierra Espuña", 2 escalera, 1 derecha, 30.008 Murcia), con más hambre que esperanzas, se decidió por la aventura pidiendome el listado en assembler de la ROM del QL.

Afortunadamente para aquellos usuarios de Amiga que deseen emular el QL, en la pasada European Microfair de octubre'89, el grupo Holandés de usuarios QL presento la primera versión de su emulador de QL en el Amiga. Desafortunadamente es bastante lento en acceso a disco, scrollando produce fantásticos colores, y olvida unos pocos puntos a la derecha de la pantalla. La principal necesidad es un acceso a disco más rápido. Los puntos en el Amiga son más estrechos que en el QL, tanto que el menu de ficheros rectangular de la QRAM parece cuadrado. Pero la gran ventaja, es que se trata de un programa de dominio público (en otras palabras, GRATIS).

P. Borman (el secretario de QUANTA) va a intentar obtener una copia para la libreria de QUANTA.

S. Merino, 19/12/1989

TEXT87 LA NUEVA VERSION 3.00 PARA LOS 90

La nueva versión posee integrado un comprobador de ortografía a 200 palabras por segundo y un diccionario de más de 40.000 palabras Inglesas, Francesas y Alemanas como Standard (Holandés e Italiano, según demanda).

El interface usuario ha sido perfeccionado.

Más flexible editando y opciones de manipulaciones de texto.

TEXT87 v3.00 cuesta 60 libras

TEXT87+founted89+fountext88 cuesta 95 libras

SOFTWARE87, 33 Savernake Road, London NW3 2JU

S. Merino, 19/12/1989

CARTAS ABIERTAS

CORREOS, LLUVIAS Y TELEFONICA

Valencia 13/12/89

Estimado Salvador:

Pasadas ya las aguas de estos dias, me he decidido a llamarte ya que hacia tiempo que no sabia nada de ti ni del CUQ. Han bastado tres intentos para contactar contigo. Lo de los intentos tiene su explicacion ya que mi telefono va fatal y lo normal es que las llamadas, caso de realizarse, terminen por imperativo del ruido de fondo que hace imposible cualquier comunicacion normal.

Como te he comentado hace un momento te envie un disco que creo no has recibido. Yo he esperado pacientemente tu respuesta pero primero la paciencia y despues el temporal de agua y las noticias que llegaban de tu zona me han hecho aguardar hasta hoy. Ahora estoy con el "mono", ya no aguanto mas y te escribo porque preciso de noticias frescas sobre el QL. Lamento no ser generador de noticias pero este año estoy muy liado con los cursos de doctorado y poco podre hacer. Por cierto uno de ellos es Bases estadisticas en Ciencias de la Salud, y si lo logro acabar podre escribir un programa estadistico de uso general y que este bien. Si alguien esta interesado en la estadistica el curso es muy util y practico, pero exige mucha dedicacion y horas.

Lo ultimo que te mande , aquel generador de aplicaciones para archive esta en ese lamentable estado en el que lo conoces, no lo he tocado. La culpa la tiene en parte el PC que tengo. El motivo es el siguiente, como no me gusta el basic de Microsft pues me he agenciado el TURBO BASIC , sobre el papel muy bien , parecido al Superbasic. En la realidad no funciona tan bien y los programas compilados no tienes certeza de que funcionen. He probado varios basic y no he encontrado nada que iguale al Superbasic + Qliberator, realmente es una suerte poder trabajar asi.

El ultimo numero que tengo del CUQ es el 13.

Saludos y animo para seguir en la brecha.

Felices fiestas y feliz y prospero año nuevo

M. Frasquet (Valencia)

Te envio los programas que se usan para pasar ficheros de el paquete de Psion de un QL a un PC. Se hace via RS323 y mediante un cable normal . En el PC debe de cargarse un programa que tengo a disposicion del que lo quiera. Asi mismo si alguien quiere el Xchange en version para PC disco de 5.25 se lo puedo enviar.

45 KBYTES + EN FORMATEAR UN DISCO

Estimados amigos sufridos usuarios del QL.

Aprovecho esta primera ocasión que tengo para ponerme en contacto a través de esta inmarcesible revista para contaros mis penas y desvelos en mi relación con el QL y presentarme. Me llamo Javier, tengo 27 años y soy Ingeniero técnico electrónico, aunque no he ejerzo coma tal en este momento, ya que desde hace un año trabajo en una empresa dedicada a la neumática.

Todavia hoy recuerdo el dia que compre mi amado QL. Despues de un eficaz lavado de cerebro para tratar de convencer a mi madre de lo útil que seria un

ordenador (entonces yo estaba todavia estudiando y no tenia ni un ochavo) lo consegui. Corria en año de gracia de 1985. De esta guisa me dirigi ilusionado al Corte Inglés, que por aquellas fechas (Noviembre) estaba en una de sus promociones de material electrónico.

Desgraciadamente, a la primavera siguiente se produjo la caida del imperio SINCLAIR y el #$%>!* de Alan Sugar firmó la sentencia de muerte de nuestro querido ordenador. A pesar de ello, y aunque en mi primer trabajo tuve oportunidad de trabajar con PCs (con MS-DOS y XENIX)y Macintosh, yo sigo siendo fiel a mi QL, y lo seguire hasta que el cuerpo (y las reparaciones) aguante.

Para ser sincero diré que el Macintosh me encanta pero su precio es muy alto. También os dire que estoy pensando muy seriamente en comprarme un ATARI y utilizar el emulador que comercializa Joachin Merz, ya que me parece una combinación muy interesante. Llegado en caso os contaria mis experiencias, ya que pienso que podría ser interesante.

Sobre C.U.Q. me parece una idea fenomenal. Aunque solo tengo en mi poder los números 11 y 12 me parece que en nivel de la misma es bastante alto, que no tiene nada que envidiar a QUANTA (del que soy miembro desde el año 88) y encima en castellano, que no deja de ser una importante ventaja para los que no dominamos del todo el inglés. Curiosamente, me enteré de la existencia de C.U.Q. a través de QUANTA, por un árticulo de Salvador Merino.

Por último y para no enrrollarme más me gustaria hacer un comentario sobre algo que he descubierto recientemente. Además de mi QL, tengo desde hace un año una placa SANDY SUPERQBOARD con 512K, y una unidad de disco de 3.5" marca SANDY (NEC), de los cuales estoy muy contento. Pues bien, en el manual de la placa, donde se refiere al tema de la interface del disco, se habla del comando FLP_TRACK, para poder formatear discos a 40 pistas (simple densidad). Leyendo la revista inglesa ST WORD, me encontre con un árticulo sobre las unidades de disco para los ATARI ST. Estos ordenadores usan unas unidades identicas a las de los QLs (bueno, a aquellas capaces de formatear discos en doble cara/doble densidad de 3.5") que son capaces de dar 1Mb sin formatear. Pues bien, en esta revista indicaba que era posible formatear estos discos hasta 948 Kb, usando 11 sectores por cara y 85 pistas.

Animado con esto se me ocurrio usar el comando FLP_track 85 y decubrí que haciendo un FORMAT FLP1_ a continuación, obtenia un disco con 1530 sectores, en lugar de los 1440 habituales. He trabajado con este disco durante un tiempo y no he tenido ningún problema.

En la mencionada revista se hablaba de que este tipo de formateos pueden no ser siempre fiables, ya que se depende mucho del mecanismo del drive en cuestion, pero que usando 10 sectores por cara y 82 pistas se pueden conseguir 830Kb de una forma fiable. Yo no se la forma de hacer un formateo a 10 sectores en el QL, pero seria interesante tener discos con 110Kb de regalo. Tal vez no sea posible, pero si hay alguien que lo sepa, que hable ahora o que calle para siempre!.

De todas formas 45Kb de más tampoco están mal, aunque no estoy del todo seguro de que sea un formateo fiable 100%, por que todavia hace poco que he descubierto esto.

Bueno, termino por fin. Un saludo y a seguir con el QL, en la esperanza de que algún dia Alan Sugar (a la sazón presidente de AMSTRAD) decida desenterrar la idea de un digno sucesor a nuestro amado y nunca bien ponderado QL.

Javier Zubieta Aguirre
Bilbao, 18 de Diciembre de 1989.

¡UNA DUDA EN ASSEMBLER!

Me gustaría que alguien me dijese por que al ejecutar este programa el ordenador hace un reset. Aparentemente, por lo menos para mí, no debería ocurrir nada extraño.

El programa es el siguiente:

```
     ;Programa para pasar la pantalla de una zona superior de la memoria

       lea      200000,a2 ; Dirección donde tengo cargada la pantalla
       lea      131072,a3 ; Dirección del comienzo de pantalla
       move.l   #0000FFFF,d4 ; Número de bytes totales de la pantalla
otro   move.l   (a2)+,(a3)+ ; Pasa contenido de a2 a a3
       subq.l   #4,d4 ; Resta 4 a d4, ya que coloca 4 bytes cada vez
       bne      otro ; Si no es cero sigue
       move.l   #0,d0 ; Según  trae la guia sirve para retornar al basic sin
                        problemas
       rts  ; Retorna.
```

La pantalla efectivamente aparece en la pantalla sin ningún problema pero una vez que está entera es cuando hace el reset.
Hasta la próxima.

Celestino Alvarez.

Por fin alguien pregunta algo sobre problemas en la escritura de un programita. Ya era hora, porque sé que todos tenemos problemas, pero nadie quiere preguntar.
Tu problema, Celestino, es muy fácil de resolver. Primero FFFF son 64 Kbytes (no entiendo como tu assembler lo ha aceptado sin el simbolo $). Y el reset viene debido a que cuando pasas de los 32 Kbytes, escribes sobre las variables del sistema.

```
     ;Programa para pasar la pantalla de una zona superior de la memoria

       lea      200000,a2 ; Dirección donde tengo cargada la pantalla
       lea      131072,a3 ; Dirección del comienzo de pantalla
       move.l   #32768,d4 ; Número de bytes totales de la pantalla
otro   move.l   (a2)+,(a3)+ ; Pasa contenido de a2 a a3
       subq.w   #4,d4 ; Resta 4 a d4, ya que coloca 4 bytes cada vez
       bne      otro ; Si no es cero sigue
       move.l   #0,d0 ; Según  trae la guia sirve para retornar al basic sin
                        problemas
       rts  ; Retorna.
```

S. Merino, 29/12/1989

MANUSCRITO PARA ESCRIBIR UN EMULADOR DE ZX SPECTRUM

Hace ya un par de años que habia comprado el emulador SUCCESS, más que nada por tener un emulador, pues el CP/M nunca me ha gustado. En el manual se decia que con alguna programación extra, podria ser usado para emular otro ordenador basado en el Z80. Como ejemplo ponian al Spectrum con un rotundo ARGGHHHH!.
Como siempre alguien ha despertado mi curiosidad (Marcos Cruz con el retorno de adapta pantallas Spectrum a QL), y me he tomado la libertad, aprovechando que llevamos en Málaga 4 semanas lloviendo y con levante fuerte en el estrecho, de repasar las instrucciones del SUCCESS y lo que me queda de información del Spectrum.
Lo primero que necesitamos es un intérprete de Z80, y lo tenemos, el mismo que usa el SUCCESS. En el manual podemos encontrar los registros 68000 conteniendo los valores de los registros del Z80.
El emulador SUCCESS una vez cargado busca en el drive flp1_ un fichero llamado BIOS_CDE, el cual es invocado cuando una instrucción IN o OUT del Z80 es usada. Ese fichero consiste en una serie de instrucciones Z80 OUT y algunos datos de necesidad en el medio Z80.
En nuestro caso particular, el Spectrum, solamente necesitamos:
- El intérprete Z80 naturalmente.
- Un listado del fichero BIOS en assembler para observarlo, pues tenemos que crear otro BIOS encargado de los ports de entrada y salida del Spectrum que se comunican con los periféricos: El TV, teclado y cassette (y el zumbador). El acceso al teclado y el cassette se realiza mediante el port 254. Pero para el TV se encarga la ULA, y en nuestro emulador seria un JOB adaptase el mapa de pantalla del Spectrum en las direcciones de la pantalla del QL cada cierto tiempo, pero con la velocidad instantanea del comando SPECLOAD.
- Necesitamos una copia de la ROM de un Spectrum que colocaremos en su posición original dentro de una página de 64K (o eso creo).
En el BIOS debe haber una rutina encargada de leer el teclado del QL comprobando si hay pulsación para devolver el valor correspodiente en la variable LAST-K del Spectrum y en el registro A (D7.B en nuestro sistema). También deben escribirse unas rutinas para hacer el LOAD y SAVE del cassette, simplemente que en nuestro caso seria FLP1_, y aqui si que hay bastante lio, pero no tanto. El problema reside en que la única información que he podido encontrar de las rutinas ROM del Spectrum es la que viene en el Código máquina por capítulos de Microhobby (otros manuales o libros no los tengo disponibles en este momento), y la información está bien para programas en código máquina, pero ¿Cuál es la diferencia para programas en BASIC? Quizás no sea muy importante, pero cualquiera sabe.

Según mi opinión, no sé a quien le puede interesar un emulador de Spectrum que dificilmente obtendrá la velocidad de un ZX 81, y va a necesitar tener pegado un ZX Spectrum con interface serie de los de verdad para pasar los programas a disco QL, porque si ya tenemos un ZX Spectrum (valorado en menos de 5.000 ptas, según cotización del mercado actual) es cosa de locos. Pero si algún dia no tengo nada que hacer, ¿Quién sabe? Es un objetivo en programación para marcar diferencias entre diferentes microprocesadores.

S. Merino, 3/12/1989.

LENGUAJE SuperBASIC

SOLUTION - CONFIG II. ERRATA CORRIGE.

En el número último se publicó un breve articulillo mío sobre el job de control de pantalla del famoso Solution, acompañado por un programita en SuperBasic que se encargaba de modificar la prioridad de dicho job a voluntad del usuario. No se si muchos habreis intentado utilizarlo: si es así es posible que hayais tenido algún problema o que hayais notado alguna cosa extraña.

Los errores que he localizado en la versión que os envié son dos: uno de poca monta, aunque algo molesto; otro menos molesto en principio, aunque absolutamente garrafal.

El primer errorse encuentra en la línea 290; donde dice

```
290 LBYTES flp1_solution,Base               debe decir

290 LBYTES source$&"solution",Base
```

Como se puede apreciar el problema se limita a que, si intentamos hacer la configuración del programa a partir de un original situado en una unidad de disco diferente de flp1_, por mucho que se lo digamos al programa de configuración no nos hará ni caso y se empeñará en buscarlo en flp1_.

El segundo error es algo más serio. Quienes hayais probado mi programa habreis notado que las copias de solution configuradas con él no funcionan mal en general (o por lo menenos no peor que el original) escepto en una cuestión: cuando aparece la pantalla de presentación los caracteres del título se ven sustituidos por cualesquier otros y en posiciones raras. La causa se encuentra en la línea 520; donde dice

```
520 SEXEC source$&"Solution_CNF",Base,Longitud,0    debe decir

520 SEXEC source$&"Solution_CNF",Base,Longitud,1024
```

El resultado de este último error es que dejamos la copia de Solution configurada sin espacio de memoria reservado para datos. El primer efecto es el que os he contado, pero a la larga se puede llegar incluso a colgar el QL, con lo que la cosa es algo más seria.

Aprovecho para mandaros la versión 1.01 de Edisc, en la que se ha corregido también un errorcillo: la versión anterior, en el caso de no poder leer un sector por encontrarse éste defectuoso, dejaba en pantalla el contenido del sector anteriormente leido; en la nueva versión, como es lógico, las ventanas de volcado ASCII y Hexadecimal se borran y se quedan así hasta que no sea leido un sector con éxito.

A parte de esta corrección estoy pensando en una nueva versión con bastantes mejoras sobre la actual: epero tener tiempo para meterme con ella.

José Carlos de Prada.

OVERDRIVE

El programa que acompaña a éste artículo lo escribí porque no conseguí que el configurador normal que acompaña al Overdrive corriera correctamente en mi QL versión JM, se quedaba colgado.

En su funcionamiento mi programa es bastante más arcaico que el

original de Overdrive, pero por lo menos funciona. A diferencia del configurador normal del Overdrive no trabaja con un fichero intermedio "_set" para luego producir la versión codificada definitiva sino que altera directamente los ficheros "_data" de la traducción. Una descripción detallada de la estructura del fichero TRANS_DATA que acompaña al Overdrive puede encontrarse en el apéndice al final de sus instrucciones, ahí es donde me he basado para escribir mi programa.

Una vez en el menú principal podemos hacer 4 cosas: cargar un fichero, alterar un fichero, grabar un fichero y salir de programa. Las opciones 1, 3 y 4 creo que están lo bastante claras como para no necesitar explicación, me centraré por tanto en la opción 2. Lo primero que nos pide al seleccionarla es el código de los caracteres para incluir gráficos en el QUILL, en el TRANS_data original del Overdrive estos códigos son "{" (123) para ON y "}" (125) para OFF. Después nos pide el código ASCII del caracter cuya traducción queremos alterar y muestra la traducción actual para dicho código. Los códigos de la traducción deben introducirse uno a uno mediante ENTER (ej. 3 ENTER 8 ENTER 65 ENTER -1 ENTER), el -1 indica al programa que hemos terminado con la traducción. Para abandonar esta opción y regresar al menú principal debemos responder -1 cuando el programa nos pregunte que caracter queremos alterar.

El Overdrive_bas necesita, al igual que su hermano mayor, las extensiones del TURBO TOOLKIT aunque se podría prescindir de ellas a costa de eliminar el error trapping (comprobación de errores). El Overdrive_bas no necesita expansion de memoria para funcionar.

Sergio Montoro Ten
Madrid, 10 de diciembre 1989

LISP

LISP

Es posible que a estas alturas mas de uno se haya planteado alguna vez el aprender algun lenguaje nuevo al margen del SuperBasic, Lo mas sencillo es haberse sentido tentado por explorar el ensamblador del 68000, sobre todo por su velocidad y por las jugosas prestaciones que ofrece el QDOS.

Estos lenguajes, como la gran mayoria, fueron concebidos con el proposito de procesar numeros, de tal manera que se pudieran ejecutar calculos matematicos con rapidez y precision. Sin embargo, casi desde el principio de la informatica, los cientificos han tratado de emular en los ordenadores los rasgos de la inteligencia humana y en particular uno de sus elementos clave: el procesamiento simbolico.

El lenguaje mas difundido hoy en en dia en el campo de la inteligencia articficial es el LISP (acronimo de Procesador de LIStas), este lenguaje fue desarrollado en el MIT por Jhon McCarthy en 1958 y es, despues del FORTRAN, el lenguaje de alto nivel mas antiguo todavia en uso.

Como su nombre indica el LISP es un lenguaje pensado principalmente para el manejo de listas, una lista no es mas que un conjunto de palabras, numeros o listas encerrados entre parentesis y separados por uno o mas espacios:

```
(12A 32 hola bb) --> 4 elementos o atomos.
(1 2 3 4 5 6 7) --> 7 atomos
(BUENO-MALO ALTO-BAJO) --> 2 atomos
(1 2 3 (3_3 3_6 3_9) 4) --> 4 atomos y una sulista
```

Todas las intrucciones del LISP estan diseñadas en torno a estas listas; asi, podemos tomar el primer elemento de una de ellas, añadirle otro, formar una lista a partir de sus elementos, suprimir un elemento de una lista, ver si un elemento pertenece a una lista, etc. El LISP es un lenguaje ideado para tratar el procesamiento simbolico, por ello carece de numeros en coma

flotante y de las habituales funciones numericas del BASIC, excepto, claroesta, las mas basicas como la suma, resta, multiplicacion y division con enteros.

Otra caracteristica notable del LISP es la recursion. En general, cuando la definicion de un termino incluye dicho termino se dice ques es recursiva. Consideremos, por ejemplo, la siguiente definicion de factorial:

a) El factorial de 1 es uno.

b) El factorial de un entero mayor que uno es dicho numero por el factorial de dicho numero menos 1, es decir: n!=n*(n-1)!

Esto tambien se puede escribir en BASIC:

```
100 DEFine FuNtion factorial(n%)
110   IF n%=1 THEN RETurn 1
120   RETurn n%*factorial(n%-1)
130 END DEFine factorial
```

Normalmente las definiciones recursivas suelen ser mas lentas y requerir mas memoria que las tradicionales de bucles, aunque son siempre mas elegantes, sencillas y faciles de entender que los bucles. La recursion no esta implementada en muchos lenguajes de programacion. He aqui factorial haciendo lo mismo con blucles:

```
100 DEFine FuNtion factorial(n%)
110   LOCal j,k
120   LET j=1
130   FOR k=1 to n%
140     LET j=j*k
150   END FOR k
160   RETurn j
170 END DEFine factorial
```

Visto esto vamos ahora a por el LISP de Metacomco, el unico disponible para el QL. Este interprete esta basado en el LISP de Acornsoft lo que ya es algo positivo a priori porque este es un dialecto LISP bastante difundido y utilizado, el paquete es aceptable en todos los sentidos excepto en dos puntos:

a) La velocidad. Los programas en LISP no se compilan y ni siquiera es producido un pseudocodigo, como en el caso de las primeras versiones del PASCAL para el QL, con lo que el LISP no viene a ser mas rapido que el interprete BASIC convencional.

b) La documentacion es catastrofica, especialmente en lo referente a la libreria de funciones que acompaña al interprete.

En lo referente a la velocidad cabe decir que este LISP, como en general cualquier version de LISP para un micro, no esta hecho con objeto de que corran programas sobre el sino mas bien como una herramienta de desarrollo de software a bajo coste. Normalmente el LISP solo se usa en un micro cuando el programador no tiene el dinero que cuesta una estacion LISP con 64 Mbytes de RAM y un procesador especializado.

La cuestion de la documentacion es otro asunto. Un paquete para "manitas" no significa una documentacion telegrafica o incompleta, pues en el manual ni siquiera aparecen todas las funciones que soporta el interprete.

Dicho todo esto, y revisadas las pegas, el LISP es algo que recomiendo fervorosamente a todos aquellos que quieran tener algo que ver con la inteligencia artificial, el procesamiento del lenguaje, y la chatarra de quinta generacion.

He aqui algunos de los libros sobre LISP, IA y Sistemas Expertos que leido y encontrado interesantes.

Inteligencia Artificial: conceptos y programas. Tim Hartnell. 1986.

Este no es un libro sobre LISP, de hecho todos los listados que contiene, y son bastantes, estan en BASIC. No se trata de una obra divulgacion seria, sino mas bien de un conjunto de programas explicados. Aunque sirve de introduccion a las posibilidades de la IA que desde luego agradara a los impacientes de ver como corre en su QL un programa capaz de aprender.

LISP. El lenguaje de la inteligencia artificial. A. A. Berk.1986.

Esta es una obra ideal para principiantes, de hecho fue mi primer libro sobre LISP. El libro es relativamente breve y a pesar de que se deja algunas cosas en el tintero es ameno y facil de comprender aunque no se entienda demasiado sobre programacion.

LISP. Introduccion al calculo simbolico. David S. Touretzky. 1986.

Tambien es una obra muy aceptable de introduccion al LISP, aunque mas extensa y densa que la anterior. Cualquiera que empieze con este libro corre el riesgo de dejarlo por aburrimiento pasadas 20 paginas, pero que nadie se desanime, a partir de ahi el texto es estupendo.

LISP. Winston & Horn. 1984

En mi opinion este es uno de los mejores libros sobre LISP que he visto. Aunque sus aproximadamente 440 paginas de texto en ingles hacen que no sea precisamente la lectura de un tebeo.

LISP for the BBC Micro. Norman & Catell. 1984.

El interes de este libro reside en ser precisamente todo lo que le falta a la documentacion de Metacomco. De el se extrajeron todas las rutinas que acompañan al LISP del QL.

A fondo: Inteligencia Artificial. Henry C. Mishkoff. 1988.

Francamente creo que cualquiera que se compre este libro tardara en leerselo 3 dias con antes de ayer. No es tan a fondo como el titulo suguiere pero sirve perfectamente para abrirnos el espectro de lo que es y lo que puede llegar a ser la inteligencia artificial de las computadoras.

A fondo: Sistemas expertos. Louis E. Frenzel. 1989.

Pertenece a la misma coleccion que el anterior y como el anterior es simplemente una introduccion sin meterse en problemas reales de programcion. De cualquier forma es ideal para empezar o simplemente para leer algo interesante y nuevo acerca de los metodos de resolucion de problemas de la IA.

Sistemas expertos. Conceptos y ejemplos. Alty y Coombs. 1985.

Este es un libro algo mas profundo que el anterior y con un mayor grado de atencion hacia los problemas reales a la hora de programar.

Principios de inteligencia artificial. Nils J. Nilsson. 1987.

Se trata de una obra seria y extensa para aquellos que deseen abordar la IA de manera seria y extensa. Su enfoque es hacia los problemas que se plantean al sentarnos delante de una pantalla y decirnos "Bueno, vamos a hacer que esto piense". Para la lectura de este libro creo es conveniente saber algo sobre programacion y sobre algun lenguaje orientado hacia el calculo simbolico, aunque el libro no utiliza ninguno en especial.

Controversia sobre mentes y maquinas. Edicion de Alan Ross Anderson. 1987.

Este libro es diferente a los otros. En primer lugar no es un libro sino una recopilacion de articulos; en segundo lugar no es algo referido a los problemas tecnicos de la IA sino mas bien a los problemas filosoficos que entraña. A veces resulta un algo dificil de seguir pero con una lectura atenta y un poco de concentracion se hace accesible a cualquier persona.

L-TOOLKIT

Si hay algo claro en esto de la informatica es que los programadores son una de las especies mas comodonas de este planeta. Solo hay que ver la

oferta de programas para el QL: 2 procesadores de texto, 3 compiladores y 876 toolkits para la programacion.

Pero por algo se empieza. Lo que viene acontinuacion es en parte una traduccion de las instrucciones del LISP de Metacomco y en parte una recopilacion de ideas tomadas de los libros citados anteriormente. He decidido traducir del manual del LISP solo el glosario de funciones, entre otras cosas porque el resto del manual carece de interes. Todas las funciones etiquetadas como "expr" son funciones definidas en LISP y no en codigo maquina, en este grupo se incluyen las extensiones del L-TOOLKIT. A este toolkit le acompaña un programa similar al Library Manager del Turbo, que extrae las rutinas deseadas del programa principal, para usar este programa es necesario disponer de las extensiones para acceso aleatorio de ficheros del TURBO TOOLKIT o, en su defecto, las equivalentes del TOOLKIT II.

Para poder usar todas las funciones del L-TOOLKIT es necesario realizar los siguientes pasos:

EXEC flp1_LISP_BOT_exe ENTER

Eso cargara el LISP y las extensiones en codigo maquina. Ahora SIN HACER NADA QUE AFECTE A LA PANTALLA hay que pulsar CTRL C, y una vez dentro del interprete teclar:

(rdf 'flp1_LISP_LTK_lsp) ENTER

Apareceran los tradicionales mensajes de evaluacion y quedaran implementadas todas las funciones del L-TOOLKIT.

Equipo de desarrollo QL LISP Introduccion

QL LISP es una version de LISP compatible con la que usa el microordenador BBC aunque ha sido extendido con el fin de aprovechar al maximo muchas de las facilidades del QL. QL LISP corre en cualquier version del QL aunque si se dispone de una expansion de memoria sera posible montar una version tipo mainframe como muchas otras que ya estan corriendo bajo el 68000.

Signos convencionales:

< >
Se usan para encerrar un texto que describe el tipo de cosa que ira entre medio. Por ejemplo: (print <nombre-de-funcion>), donde <nombre-de-funcion> debe ser reemplazado por el valor real de la funcion a ser impresa.

[ ]
Los corchetes cuadrados se usan para rodear cualquier cosa opcional.

{ }
Las llaves encierran un tipo de parametro que no es frecuentemente usado.

a_lista
Una lista en la que cada miembro es un par-con-punto.
Ej. ((a . b) (c . d) ...)

atomo
Un atomo es cualquier tipo de numero, caracter o id.

booleana
Cualquier combinacion de la expresiones t y nil.

byte
Un numero entero con un rango comprendido entre 0 y 255.

Expr
Una funcion es expr si esta implementada en LISP.

Fsubr
Una funcion es fsubr si esta implementada en codigo maquina y trata sus argumentos de manera especial.

grabage collection
Es la operacion que realiza el LISP para poder reutilizar la celdas que se usaron y han quedado ahora libres.

id
Euivalente al atomo normal es LISP pero con un valor y una lista de propiedades asociados. Los numero son tratados de manera especial ya que no necesitan todo este mecanismo para funcionar. Por tanto id no puede ser un numero.

lista
Son grupos de atomos o lista encerrados entre parentesis.

numero
Un numero entero con un rango comprendido entre -536870909 y +536870909

par-con-punto
El objeto no atomico fundamental en LISP. Un para con punto tiene 2 componentes, llamados car y cdr. El par con punto cuyo car es "a" y cuyo cdr es "b" se escribe (a . b).

Subr
Una funcion es subr si esta implementada en codigo maquina y trata sus argumentos de manera normal.

Equipo de desarrollo QL LISP   Funciones y variables

.
El caracter . se usa para la notacion de entrada de listas, si a y b son cualquier estructura, (a . b) representa un par con punto cuyo car es a y cuyo cdr es b. Para usar el atomo '.' ver las entradas bajo ! y period.

(
Los parentesis se usan en las entradas de LISP para formar listas. Para usar el atomo '(' ver las entradas bajo ! y lpar.

!
! es el caracter escape, provoca que el siguiente caracter sea tratado como una letra ordinaria. Esto significa que los caracteres con propiedades especiales, como '(' o '.', puden ser usados como partes de un identificador.

---

*, **, ***   Variables
Los tres resultados mas recientes producidos por LISP son almacenados en la variables *, **, ***. Cada vez que el usuario interacciona con LISP estas variables son actualizadas. Esto facilita el re-usar cantidades recientemente computadas, por ejemplo:

```
'(a b c)
```

```
     (append * *)                  => (a b c a b c)
     (car **)                      => a
```

Los valores almacenados pueden ser descartados si LISP se queda sin espacio libre. Ver -, +, ++, +++.

-, +, ++, +++ Variables

Las expresiones de entrada recientemente representadas se almacenan en estas variables: - es la estructura corriente que esta siendo evaluada, mientras +, ++ y +++ son las previas a esa. Estos valores son la mayoria de las veces utiles como recordatorio de lo que fue tecleado, pero pueden servir para evitar el re-teclear largas expresiones. Ejemplos:

```
     (cdt '(a b c))                (en lugar de cdr)
     <error>
     (subst 'cdr 'cdt +)           => (cdr (quote (a b c)))
     (eval *)                      => (b c)
```

(abs U:numero) : numero Expr

Abs es una funcion que devuelve el valor absoluto de un numero.

(add1 U:numero) : numero Subr

Devuelve su argumento numerico incrementado en una unidad. Es equivalente a (PLUS U 1) pero mas rapido. Ver tambien sub1.

(adval {}) : no implementada Subr

(allocate U:numero) : numero Expr

Llama: tk c/m, send, recive.

Allocate es una funcion que reserva una determinada cantidad de memoria en el area comun. U es el numero de bytes a reservar y el valor devuelto es la direccion de inicio del area reservada. Ver apendice A.

(and [U:cualquiera]) : booleana Fsubr

Evalua cada U hasta encontrar un valor nil o el final de la lista. Si el ultimo valor no es nil la funcion devuelve ese valor, en caso contrario devuelve nil. De modo que el valor sera tratado por LISP como true solo si todos y cada uno de los argumentos no son nil. And no evalua necesariamente todos los argumentos. Recorre U evaluando los argumentos uno a uno hasta que:

i) El valor de un argumento es nil. Es valor retornado en ese caso es nil.

ii) El final de la lista de argumentos es alcanzado. En este caso el valor retornado es el del ultimo argumento evaluado (no-nil).

Por ejemplo,

```
     ( and (numberp n) (greaterp n 0) (lessp n 7) )
```

devolvera t cuando la variable 'n' sea un numero comprendido entre 0 y 7. Ver tambien or, not, t y nil.

(append U:lista V:lista) : lista Subr

Si U y V son dos listas, entonces (append U V) es la lista que se obtiene al poner todos los elementos de V detras de los de U. Por tanto (append '(p q) '(r s)) sera la lista (p q r s). Append se podria haber definido en LISP como

```
     (defun append (a b)
        (cond ((null a) b)
        ( t (cons (car a) (append (cdr a b)))))
```

pero esta implementada en LISP en codigo maquina.

(apply FN:{id funcion} ARGS:lista) : cualquiera Subr

FN debe ser una funcion en forma de un apuntador a un codigo o una expresion lambda o un identificador que haya sido definido como una funcion. ARGS debe ser una lista de argumentos de forma que esten

listos para ser asignados como parametros formales de FN (ej. si FN espera argumentos evaluados entonces los argumentos que se le pasen deben haber sido previamente evaluados). El resultado de evaluar FN con los argumentos suministrados en ARGS es el valor devuelto.

(assoc U:cualquiera V:a_lista) : {par_con_punto nil} Subr
Si U es la porcion car de un elemnto de la lista V el par con punto donde se ha hallado U es devuelto sino la funcion devuelve nil.

```
     (defun assoc (a l)
        (cond ((null l) nil)
        (equal a (caar l)) (car l))
        (t (assoc a (cdr l)))))
```

(atom U:cualquiera) : booleana Subr
Devuelve t si U es un atomo; ej. un identificador, un numero o referencia al codigo maquina. Si atom es true entonces el uso de car y cdr con U sera ilegal aunque U sea nil.

(band U:numero [U:numero] : numero Fsubr
Band trata todos sus argumentos como si fueran cantidades binarias de 28 bits y hace un and de ellas en forma binaria. Por ejemplo,

```
     (band 5 9) =1                (binario: 0101 & 1001 = 0001)
```

Ver tambien bor y bnot.

(biggerp U:atomo V:atomo) : booleana Expr
Biggerp es una funcion similar a greaterp solo que en lugar de comparar atomos compara palabras. Se sigue un orden de comparacion alfabetico, de modo que 'abc es mayor que 'bcd. Si dos palabras son iguales en todas sus letras la mayor sera la que tenga un menor numero de caracteres. Por ejemplo,

```
     (biggerp 'indio 'apache) = nil
     (biggerp 'abc 'abca)     = t
     (biggerp 'ql 'ql)        = nil
```

(bin U:numero L:numero) : lista Expr
Llama: fill$, length, ncons.
Bin es una funcion que toma un numero entero y devuelve su equivalente binario de L bits de longuitud. El valor devuelto es una lista en la que cada atomo es un digito binario. Si el numero es negativo el numero se representara en formato de complemento a dos.

---

blank Variable
El atomo blank tiene el valor inicial del caracter 32 (espacio). Para testear si ch es un espacio es posible utilizar bien (eq ch blank) o bien (eq ch (quote !)). Ver la entrada ! para mayor explicacion de lo anterior.

(bnot U:numero) : numero Subr
Bnot trata su argumento como un numero binario de 28 bits y hace el complemento a dos de dicho numero. El esquema resultante de bits es usado como el valor numerico devuelto por bnot. La representacion usada por LISP para numeros implica que que para cada numero 'n' (bnot n) tiene el mismo valor que (sub1 (minus n)). Ver tambien band y bor.

(bor U:numero [U:numero]) : numero Fsubr
Bor es similar a band, excepto que hace un or inclusivo de todos los bits de los numeros que forman sus argumentos. Asi que

```
     (bor 12 6) = 14              (binario: 1100 | 0110 = 1110)
```

(call U:numero) : 1 Expr
Llama: tk c/m, send.

Esta forma de call es identica a la que usa el interprete BASIC excepto que no soporta la alteracion directa de los registros al ser llamada. Esta funcion no esta implementada en la version 1.0 de LISP pero puede ser usada en el L-TOOLKIT haciendo uso del programa LISP_BOT_exe. Ver apendice A para mas informacion.

(car U:par-con-punto) : cualquiera Subr

car(cons a b) ==> a. Devuelve la parte izquierda de U. Se produce un error si U es un atomo o nil.

```
     (car '(tres mil hamburgesas)) = tres
```

(cdr U:par-con-punto) : cualquiera Subr

cdr(cons a b) ==> b. Devuelve la parte derecha de U. Se produce un error si U es un atomo o nil.

```
     (cdr '(tres mil hamburgesas)) = (mil hamburgesas)
```

(character N:byte) : id Subr

El argumento N debe de ser un entero entre 0 y 255. Character trata a N como el codigo ASCII de un caracter. Su valor es el de un identificador que tiene por nombre este unico caracter.

(charp U:cualquiera) : booleana Subr

Charp devuelve t si su argumento es un identificador. De otro modo devuelve nil. Por tanto charp puede ser usado para distinguir identificadores (que a veces aparecen como atomos de un solo caracter) de otros tipos de objetos en LISP, ej. numeros, apuntadores y listas. Por ejemplo:

```
     (charp 'abracadabra) = t
     (charp 42)           = nil
     (charp (cons a b))   = nil
```

(chars U:cualquiera) : numero Subr

Chars devuelve el numero de caracteres que aparecerian en la pantalla si el argmento fuera imprimido. Por ejemplo,

```
     (chars 'indio) = 5
```

(circle RADIO:numero) : 0 Subr

Circle dibuja un circulo con el radio especificado en la posicion actual de la tortuga.

(circleat X:numero Y:numero RADIO:numero) : 0 Subr

Circleat dibuja un circulo con centro en la coordenadas X e Y con el radio especificado. Es equivalente a moveto + circle.

(clock): lista Subr

Esta funcion devuelve una lista de tres numeros que representan el tiempo, en horas, minutos y segundos, que ha transcurrido desde que la computadora fue reseteada por ultima vez. Ver time, getime y reset.

(close FICHERO:cualquiera) : cualquiera Subr

Close cierra el fichero con el nombre indicado y borra todos los buffers asociados a el. Devuelve error si el fichero no puede ser cerrado.

(cls) : nil Subr

Borra la pantalla.

(concat NOMBRE1:id NOMBRE2:id) id Subr

Esta funcion crea un identificador cuyo nombre es el resultado de concadenar NOMBRE1 y NOMBRE2. Es un ejemplo de como los identificadores

pueden ser usados como soporte de algun tipo de manipulacion de cadenas

---

(cond [U:forma-condicional]) : cualquiera Fsubr
Una forma-condicional es una lista de la forma (predicado expresion ... expresion). El predicado de cada U es evaluado hasta que un valor no-nil es encontrado. La secuencia de expresiones que sigue a este predicado son evaluadas y en valor de la ultima sera el valor de cond. Si todos los predicados evaluan a nil el valor de cond es nil y si no hay expresiones que sigan a un predicado el valor devuelto es el de ese predicado.

(cons U:cualquiera V:cualquiera) : par-con-punto Subr
Devuelve un par-con-punto que no es eq a nada preexistente y tiene U como car y V como cdr.

(consr U:lista V:cualquiera) : lista Expr
Consr toma la lista U y añade V como ultimo elemento de ella. Por ejemplo,

```
     (consr '(Soy un) 'asesino) =>
     (Soy un asesino)
     (consr '(Soy un) '(asesino sanginario)) =>
     (Soy un (Asesino sanguinario))
```

---

cr Variable
El valor de cr is el identificador cuyo nombre es un retorno del carro. Por tanto (princ cr) es equivalente a (print). cr = (character 10). Ver tambien blank.

(cxxxr U:lista) : cualquiera Subr
Cualquier nombre de la forma cxxxr, donde las equis representan bien carateres 'a' o 'd', es tratado como una combinacion de las funciones basicas car y cdr. Por tanto (caddr U) sera equivalente a (car (cdr (cdr U))). El maximo de letras implementadas es tres.

(defun NOMBRE:id PARAM:{id id-lista} FN:cualquiera) : id Fsubr
La funcion FN con los parametros especificados por PARAM es añadida con el nombre especificado al conjunto de funciones definidas. Cualquier definicion previa de una funcion con el mismo nombre se perdera. Ninguno de los argumentos de defun se evalua. Usar defun es equivalente a

```
     (setq nombre-funcion '(lambda parametros cuerpo ... ))
```

El valor retornado por defun es el nombre de la funcion que ha sido defindida. El segundo argumento (PARAMetros) es una lista de los argumentos y variables locales de la funcion. Cualquier numero de acciones pueden ser ejecutadas mediante una funcion.

```
Ej.  (defun factorial (n (local1 local2))
        (SETQ local1 1)
        (SETQ local2 1)
        (LOOP (UNTIL (EQUAL local2 n))
           (SETQ local1 (TIMES local1 local2))
           (SETQ local2 (ADD1 local2)) ) )
```

(deallocate U:numero) : numero Expr
Llama: tk c/m, send.
Esta funcion libera un area reservada mediante allocate. U es la direccion de inicio del area a liberar. No se deben liberar areas que se hayan usado para contener extensiones al SuperBasic. Ver apendice A para mas informacion.

(dec U:lista) : numero Expr
Dec es una funcion que toma una lista representando un numero binario y devuelve su equivalente decimal. El formato de dicha lista debe ser

identico al que produciria la funcion bin al pasar ese mismo decimal a binario.

(delete U:cualquiera V:lista) : lista Subr
Borra el primer elemento en el nivlel superior de la lista V que sea igual a U y devuelve dicha lista. Ver tambien wdelete.

(difference U:numero V:numero) : numero Subr
Devuelve U - V.

(digit U:cualquiera) : booleana Subr
Devuelve t si U es un digito, de otro modo nil. Al loro que un digito es un caracter y no un numero. (ej. con U=!2 devuelve t pero con U=2 devuelve nil).

(displace U:par-con-punto V:par-con-punto) : par-con-punto Expr
Displace es una combinacion de rplaca y rplacd. Reemplaza el car de U con el car de V y el cdr de U con el cdr de V. Por ejemplo,

```
(setq x '(a b c)) = (a b c)
(setq y '(1 2 3)) = (1 2 3)
(displace (cdr x) (cdr y)) = (2 3) <==> x = (a 2 3) ; y = (1 2 3)
```

---

dollar Variable
El valor inicial de dollar es el caracter $.

(draw U:numero) : 0 Subr
Draw avanza la tortuga grafica con la pluma bajada U pixels. Ver tambien move.

(drawto X:numero Y:numero) : 0 Subr
Drawto dibuja una linea de puntos entre la posicion actual del cursor y las coordenadas absolutas de pantalla X e Y. Ver draw y moveto.

(edit FN:id) : cualquiera Fsubr
La funcion edit no evalua su argumento si que este no necesita ser entrecomillado. Edit hace una impresion "legible" de la definicion asociada al identificador pasado y luego usa set par reemplazar esa misma definicion con lo que sed devuelva. Edit podria haber sido definida en LISP como

```
(defun edit (name)
   (superprint (eval (car name)))
   (set (car name) (sed (eval (car name))))
   (terpri)
   (car name))
```

Los comandos aceptados en el LISP version 1.0 son:

```
A       Mover a la parte car.
B       Mover un nivel atras.
C s     Insertar expresion s con cons al principio de la lista actual.
D       Mover a la parte cdr.
R s     Reemplazar la expresion actual con s
X       Excinde la cabeza de la lista corriente
<ENTER> Realiza una impresion legible de la expresion actual.
```

(envelope {}) : no implementada Subr

(eof FICHERO:numero) : booleana Subr
Eof detecta si se ha alcanzado una marca de fin de fichero mientras se leia el mismo. Devuelve t es ese caso y nil si el fin-de-fichero no se ha alcanzado. El argumento FICHERO es un identificador de fichero obtenido a traves de open.

(eq U:cualquiera V:cualquiera) : booleana Subr

Devuelve t en los siguientes casos:
i) U y V son el mismo identificador.
ii) U y V son numeros iguales.
iii) U y V son dos lista identicas.
En cualquier otro caso devuelve nil.

(equal U:cualquiera V:cualquiera) : booleana Subr

Devuelve t si U y V son iguales. Los pares con punto son computados de forma recursiva desde el nivel inferior de sus arboles. Los apuntadores de funcion deben tener valores eq.

(error [MENSAJE:cualquiera]) : cualquiera Subr

Error se comporta como print en cuanto a que muestra en pantalla su argumento MENSAJE. Habiendo hecho eso genera el error numero 15 y la secuencia habitual de error se produce. He aqui un ejemplo de su uso chequeando que 'w' es una lista antes de intentar hallar su cdr:

```
     (cond ((atom w) (error (list w blank 'not 'list)))
           (t (cdr w)))
```

(errorgen U:numero) : lista Expr

Si el valor de U es un numero negativo entre -1 y -21 errorgen devolvera el mensaje de error standart asociado a dicho codigo, en cualquier otro caso errorgen devolvera nil.

(errorset U:cualquiera FLAG:entero) : cualquiera Fsubr

Normalmente cuando ocurre un error evaluando una expresion el rastreo regresivo (backtrace) trabaja sobre todas las llamadas a funciones y detiene el programa. Errorset es una forma de evitar esto y mantener el control del programa. El argumento pasado a errorset es una funcion para ser evaluada y que puede fallar. Si la evaluacion de esta funcion es satisfactoria errorset actua simplemente como list -ej. (errorset <expresion>) es equivalente a (list <expresion>)-. Tenga en cuenta que en este caso el valor devuelto por errorset nunca es un atomo. Si la evaluacion de la expresion falla errorset devuelve el valor del error producido. Por tanto el siguiente bucle devolvera una expresion leida del teclado pero detectara los errores que podrian ser provocados en read por parentesis o puntos descolocados.

```
     (loop (setq x (errorset (read)))
           (until (listp x) (car x))
           (print '(pruebe a teclear otra vez por favor)))

     (errorset (car nil)) => 14
```

y un mensaje de error controlado por messon/messoff

```
     (errorset (cons 'a 'b)) => ((a . b))
```

Ver messon y messoff para el control sobre la cantidad de informacion sobre diagnosticos impresa cuando ocurre un error.

(eval U:cualquiera) : cualquiera Subr

U es evaluada como una fraccion de codigo LISP con respecto a la coleccion actual de variables ligadas. Eval hace casi todo el trabajo de evaluacion de expresiones en LISP.

(every U:predicado V:lista) : booleana Expr

Llama: consr, ncons.
Devuelve t si todos los elementos de la lista V satisfacen el predicado dado, en caso contrario devuelve nil.

```
Ej.  (every 'numberp '(1 2 3 4)) => t
     (every '(lambda (x) (greaterp x 2)) '(1 2 3 4 5)) => nil
```

```
(explode U:cualquiera) : id-lista                         Subr
El  valor  devuelto  por esta funcion es una lista formada por atomos de
un  unico  caracter  que  corresponden a los caracteres originales de U.
Por ejemplo:
     (explode 'sinclair) => (s i n c l a i r)
Ver tambien implode.
```

```
f                                              Identificador especial
El  valor  inicial  de f es nil. Si f es usado como sinonimo de nil debe
evitarse su uso como variable.
```

```
(fill U:booleana) : booleana                              Subr
Fill  activa  o  desactiva  el  modo  fill (rellenado) de pantalla, t lo
activa, nil lo desactiva.
```

```
(fill$ U:cualquiera V:numero) : lista                     Expr
Llama: consr.
Fill$  toma  el  valor  de U y devuelve una lista resultado de repetir V
veces U. Por ejemplo:
     (fill$ '* 5) => (* * * * *)
```

```
(find-if U:cualquiera V:lista) : cualquiera               Expr
Llama: consr, ncons.
La  funcion  find-if  va  aplicando sucesivamente el predicado U a todos
los  elementos  de  V.  Devolvera  el  primer  elemento que satisfaga el
predicado o nil si no encuentra ninguno que lo haga. Ej.
     (find-if 'zerop '(3 2 1 0 -1 -2 -3)) => 0
```

```
(flatten U:lista) : lista                                 Subr
Flatten  toma  una  lista  y devuelve una lista de un solo nivel formada
por todos los atomos encontrados en la lista tomada como argumento.
```

```
(freeze) : numero                                         Expr
Frreze es una forma de emular el bloque de pantalla de CTRL-F5.
```

```
(fsubrp U:cualquiera) : booleana                          Subr
Fsubrp  testea  si  su  argumento  es  un  atomo  de tipo Fsubr, es caso
afirmativo  devuelve  t, sino devuelve nil. Los atomos Fsubr representan
los  puntos  de  entrada  de  aquellas  funciones  LISP que procesan sus
argumentos de una manera especial. Por tanto
     (fsubrp cond)  => t
     (fsubrp cons)  => nil
     (fsubrp 'cond) => nil
El  ultimo  caso  devuelve  nil  porque su argumento es el identificador
cond que no es lo mismo que el apuntador cond. Ver tambien subrp
```

```
(get U:cualquiera IND:cualquiera) : cualquiera            Subr
Devuelve  la  propiedad  asociada  con  el  indicador IND en la lista de
propiedades  de  U.  Devuelve nil si U o IND no son identificadores. Get
no puede ser usado para acceder a funciones.
```

```
(getchar F:fichero) : id                                  Subr
Devuelve  un  identificador  de un solo caracter. Este caracter es leido
del  teclado.  Es  posible  pasar un identificador de fichero (ver open)
como  argumento de getchar, en ese caso el caracter se leera del fichero
especificado. Ver tambien readline, read y ordinal.
```

```
(getime) : entero                                         Subr
El  valor  devuelto  por getime es la cantidad de tiempo (en unidades de
una  centesima  de  segundo)  que  se ha empleado en realizar el grabage
```

collection. Esta cantidad de tiempo es puesta a cero por reset. Ver tambien time.

(greaterp U:numero V:numero) : booleana Subr
Devuelve t si U es mayor que V, en cualquier otro caso devuelve nil.

(head U:numero V:lista) : cualquiera Expr
La funcion head toma un numero y devuelve el elemento asociado en el primer nivel de una lista. Por ejemplo:

```
     (head 3 '(a b c d e)) => c
     (head 2 '(1 (1_2 1_4 1_6 1_8) 2 3)) => (1_2 1_4 1_6 1_8)
```

(home) : 0 Subr
Home es equivalente a (moveto 500 500) (turnto 0).

(implode U:cualquiera) : id-lista Subr
En argumeto pasado a implode debe ser una lista de identificadores, donde cada item de la lista sea un atomo de un unico caracter. Implode devuelve el identificador cuyo nombre consiste en estos caracteres. El resultado de implode es un identificador incluso si todos los caracteres de U son digitos e incluso si existen signos de puntuacion (tales como parentesis, espacios en blanco, etc.) La funcion inversa de implode es explode. Ejemplos:

```
     (implode '(c a r))              => car
     (implode (cdr (explode 'that))) =>  hat
```

Ver tambien numob.

(ink U:byte) : color Subr
Cambia el color de la tinta en pantalla.

(intersection U:lista V:lista) : lista Expr
Llama: consr.
Realiza la interseccion de dos listas y devuelve otra lista formada por los elementos comunes a U y V.

```
     (intersection '(a s d f g) '(v w s r a)) => (a s)
```

(keyrow FILA:numero) : numero Expr
Llama: tk c/m.
Keyrow devolvera un numero que indica la tecla que esta siendo pulsada en ese mismo instante en la FILA especificada. Keyrow puede utilizarse para leer dos teclas a la vez pero solo si estas estan en una fila diferente. Los codigos de fila y columna para cada tecla pueden encontrarse en la guia del usuario que acompaña al QL.

---

lambda Identificador especial
Lambda es un atomo que indentifica una parte de codigo LISP como representacion de una funcion. La sintaxis correcta para su uso es

```
     (lambda (variables) <expr1> <expr2> ... <exprn>)
```

(last U:lista) : cualquiera Subr
Devuelve el ultimo elemento de la lista U.

(lbytes FICHERO:id DIR:numero) : nil Expr
Lbytes carga un FICHERO en memoria a partir de la DIReccion especificada.

(length U:cualquiera) : numero Expr
Length devuelve el numero de elementos que componen una lista en su primer nivel. Si U es un atomo length devuelve 1. Por ejemplo:

```
     (length '(a (b c) d e)) => 4
```

(lessp U:numero V:numero) : booleana Subr
Devuelve t si U es menor que V. En caso contrario devuelve nil.

---

linewith Variable
El valor de linewith es establecido por el sistema y refleja la anchura de la ventana actual. Linewith es reseteado cuando se llama a window.

---

link Variable
Esta es una variable que no existe en el LISP standart. Es creada al inicializar el L-TOOLKIT y su valor es el de la base del area reservada de memoria que sirve de comunicacion entre el LISP y el LISP_BOT_exe. Su valor inicial es POKEado por el LISP_BOT en la direccion de memoria (larga) 131072 antes de inicializar el interprete LISP. Entonces el L-TOOLKIT lee esta posicion de memoria y asigna su valor a la variable link. Link no debe ser alterada o las funciones implementadas en torno al LISP_BOT no funcionaran.

(list [U:cualquiera]) : lista Fsubr
Devuelve una lista formada por la evaluacion de cada elemento de U.
Ej. (list 'a 'b 'c) => (a b c)

(listp U:cualquiera) : booleana Subr
Listp devuelve t si su argumento es una lista o un par con punto, devuelve nil si U es un atomo o es nil. Listp es la funcion complementaria de atomp.

(load U:nombre-fichero) : Subr
El argumento de load debe ser el nombre de un fichero creado mediante save. Load lee el fichero y lo carga en memoria devoviendo todo el espacio de trabajo del LISP al estado en que estaba cuando se realizo el save. Load destruye todas las definiciones de variables y funciones actuales y las sustituye por las grabadas en el fichero.

(loop U:accion [V:accion])) : booleana Fsubr
Esta funcion se usa conjuntamente con until y while. Su cometido es la ejecucion repetitiva de una conjunto V de operaciones en LISP. Por ejemplo,

```
     (setq c 0)
     (loop (until (EQUAL c 10)) (print c) (setq c (add1 c)))
```

imprimira todos los numero enteros desde 0 hasta 10.

---

lpar Variable
El valor inicial de lpar es el atomo '(', el perentesis izquierdo.

(map FN;funcion X:lista): cualquiera Subr
Aplica la funcion FN a los sucesivos segmentos cdr de X -ej. X, (cdr X), (cddr X)...- Podria haber sido definida en LISP como:

```
     (defun map (fn l)
        (cond ((null l) nil)
              (t (cons (fn l) (map fn (cdr l))))))
```

(mapc FN:funcion X:lista): cualquiera Subr
Aplica FN a los sucesivos segmentos car de la lista X -ej. (car X), (cadr X), (caddr X)...- Podria haber sido definida en LISP como:

```
     (defun mapc (fn l)
        (cond ((null l) nil)
              (t (cons (fn (car l)) (mapc fn (cdr l))))))
```

(member A:cualquiera B:lista) : booleana Subr

Devuelve nil si A no es miembro de la lista B, en caso contrario devuelve la lista de elementos de B cuyo primer elemento es A.

(messoff U:byte) : byte Subr

Messon y messoff se usan para controlar si determinados mensajes se imprimen o no. Messon permitira que el mensaje aparezca en pantalla y messoff lo suprimira. Una vez que el status de un mensaje ha sido asignado de esta manera permanece inalterable a no ser que ocurra un error auntenticamente catastrofico o se evalue otro messon/off. Cada grupo de mensajes esta relacionado con un bit de U. Por tanto, los mensajes se controlan de la siguiente forma:

Numero	Mensaje
1	Bytes de grabage collection recuperados para su uso
2	Numero de grabage collection
4	Numero de error
8	Argumentos con error del nivel superior (top level)
16	Error backtrace
128	Indicador de profundidad de read

Por tanto (messoff 16) suprime los detalles sobre los errores encontrados en el backtrace hasta que se indique lo contrario, (messoff 3) desconectara todos los mensajes sobre el grabaje collection y (messon 128) activa los indicadores '>'. En control que estas funciones proporcionan sobre los mensajes puede ser util utilizado conjuntamente con errorset.

(messon U:byte) : byte Subr

Ver messoff.

(minus U:numero) : numero Subr

Minus niega su argumento. La substraccion se puede realizar con difference.

(minusp U:numero) : booleana Subr

Devuelve t si U es un numero negativo, en caso contrario devuelve nil.

(mode U:numero) : 16 Subr

U debe ser 0 u 8: resetea el modo de pantalla para que soporte 4 u 8 colores.

(move U:numero) : nil Subr

Move mueve la tortuga grafica con la pluma levantada U pixels adelante. Ver tambien draw.

(moveto X:numero Y:numero) : nil Subr

Moveto mueve la tortuga a las coordenadas absolutas de pantalla indicadas por X e Y con la pluma levantada. Ver tambien drawto y move.

(nconc U:lista V:lista) : lista Expr

Nconc es una version destructiva de append. Mientras que append crea una lista nueva para su resultado, nconc cambia fisicamente la ultima celda cons de su primer argumento para que apunte al segundo. Ejemplo:

```
(setq l '(a b c))
(setq n '(1 2 3))
(append l n) => (a b c 1 2 3)  ;  l = (a b c)       ; n = (1 2 3)
(nconc l n) => (a b c 1 2 3)   ;  l = (a b c 1 2 3) ; n = (1 2 3)
```

(ncons U:cualquiera) : par-con-punto Expr

Ncons es una funcion que sirve para convertir atomos en listas. Es equivalente a (cons U nil).

---

nil Indentificador especial
Nil es un identificador que LISP usa de muy distintas maneras. Por tanto no es posible usar nil como nombre de una funcion o de una variable.
El primer uso especial de nil es que todas las listas terminan con un apuntador al atomo nil de modo que (a b c) es en realidad (a b c . nil) Esto afecta al programador de a pie en que el chequeo null puede ser usado para comprobar si se a alcanzado el final de una lista.
El segundo uso especial de nil es el ser la denotacion standart para 'falso' en LISP. Todos los predicados en LISP devolveran nil para falso o cualquier otro valor para verdadero.
Nil se usa con tanto frecuencia en LISP que ademas de ser un valor es una variable con su propio valor, de modo que es posible escribir (cons a nil) en vez de (cons a (quote nil)).

(not U:cualquiera): booleana Subr
Si U es nil devuelve t, en caso contrario devuelve nil.

(nth U:numero V:lista) : cualquiera Expr
Llama: nthcdr.
Es equivalente a (car (nthcdr U V)).

(nthcdr U:numero V:lista) : cualquiera Expr
Nthcdr toma un numero (U) y una lista (V) y devuelve el n-simo cdr de la lista. Por ejemplo:

```
(nthcdr 2 '(a b c d)) => (c d)
```

(null U:cualquiera) : booleana Subr
Devuelve t si U es nil. Es equivalente a (EQ U nil). Ver tambien not.

(numberp U:cualquiera) : booleana Subr
Devuelve t si U es un numero, de otro modo devuelve nil.

(numob U:lista-de-ids) : numero Subr
Numob es similar a implode pero el valor retornado por numob es el numero decimal que tiene como valor la secuencia de digitos de U.

(oblist) : lista-de-ids Subr
Devuelve una lista formada por todos los identificadores conocidos actualmente en LISP excepto aquellos que tengan el valor undefined y/o aquellos que tengan su lista de propiedades vacia. Estas condiciones eliminan aquellos atomos que esten siendo usados como cadenas de caracteres y no como atomos con un valor interesante. Oblist proporciona una informacion definitiva acerca de que funciones estan disponibles en un determinado momento.

(onep U:cualquiera) : booleana Subr
Devuelve t si U es el numero 1. No se produce ningun error si el argumento no es numerico. Es equivalente a (EQ U 1). Ver tambien zerop.

(op U:lista) : numero Expr
Op es una funcion que toma el valor numerico de car de U y el caddr de U y devuelve un numero resultado de operar ambos valores. Las operaciones soportadas son la adicion, +, la sustraccion, -, la multiplicacion, *, y la division, /, de enteros. Por ejemplo,

```
(op '(3 * 5)) => 15
(op '((4 + 8) / 6) => 2
```

(open FICHERO:cualquiera MODO:id) : entero (NUMERO-FICHERO) Subr
Open abre un fichero, de nombre FICHERO, para entrada o salida. Si MODO es nil un nuevo fichero es creado, en caso contrario se asume que este fichero ya existe. El valor de open es el numero que identificara ese fichero de ahi en adelante (un entero pequeño) y que sera usado en funciones como readline, write y close.

(or [U:cualquiera]) : cualquiera Fsubr
U es cualquier numero de expresiones que son evaluadas en orden segun su aparicion. Si el valor de todas es nil el valor devuelto es nil, en caso contrario el valor devuelto es el de la primera expresion evaluada que no era nil.

(order U:lista V:cualquiera) : lista Expr
Llama: length, head, nth, prelist.
Order es una funcion que se utiliza para ordenar la lista U segun la condicion de ordenacion V. Por ejemplo:

```
(order '(1 4 2 3 6 0 5) 'greaterp) => (6 5 4 3 2 1 0)
```

Debe de tenerse en cuenta que la condicion de ordenacion ha de comparar dos elementos de la lista y decidir cual va primero. Por tanto, no son validas condiciones como zerop, onep, etc. Lo que hace la funcion para reorganizar la lista en comparar uno a uno sus elementos, de modo que si '6' es mayor que '5' lo coloca primero en la lista de nuestro ejemplo.

(ordinal U:id) : byte Subr
Ordinal devuelve el codigo ASCII asociado al primer caracter de su argumento. Ver tambien character.

(peek DIRECCION:numero) : byte Subr
Peek devuelve un numero que representa el contenido de un direccion de memoria. DIRECCION, por tanto, debe ser un entero positivo. Peek debe ser usado con cuidado ya que las variables pueden ocupar diferentes posiciones de memoria en diferentes sistemas.

(peek_w DIRECCION:numero-par) : numero Expr
Llama: bin, dec.
Peek_w devuelve un numero que representa el contenido de una palabra en la memoria. DIRECCION, por tanto, debe ser un entero positivo y par. Peek_w debe ser usado con cuidado ya que las variables pueden ocupar diferentes posiciones de memoria en diferentes sistemas.

---

period Variable
El valor inicial de period es el atomo ".".

(plist U:id) : a_lista Subr
Plist devuelve la lista de propiedades asociadas a su argumento, U. Si no existe ninguna lista de propiedades asociada a U plist devuelve nil. Las propiedades deben ser normalmente accedidas usando put y get pero plist tambien puede ser util en algunos casos. Las listas de propiedades en LISP estan formadas por pares con punto con el siguiente formato: ((nombre-propiedad . valor) (nombre-propiedad . valor) ...).

(plus [U:numero]) : numero Fsubr
Plus devuelve la suma de todos sus argumentos.

(point X:numero y:numero) : 0 Subr
Point dibuja un punto en las coordenadas absolutas de pantalla indicadas por X e Y.

(poke DIRECCION:numero V:byte) : byte Subr
Poke almacena el byte V en la direccion de memoria indicada por DIRECCION. El valor devuelto por poke si no suproduce ningun error es V. Poke es una instruccion que puede corromper las estructuras internas de LISP y de la memoria; por eso debe usarse con mucho cuida- do. Ver tambien poke_w, peek y peek_w.

(poke_w DIRECCION:numero-par V:numero) : numero Expr
Llama: bin, dec, nthcdr, prelist.
Poke almacena la palabra V en la direccion de memoria indica por DIRECCION, de modo que direccion debe ser un numero par y V un entero comprendido entre -32767 y +32767. El valor devuelto por poke_w si no su produce ningun error es V. Poke_w es una instruccion que puede corromper las estructuras internas de LISP y de la memoria; por eso de- be usarse con mucho cuidado.

(prelist U:numero V:lista) : lista Expr
Prelist devuelve los U-1 primeros elementos de la lista V. Por ejemplo:

```
(prelist 4 '(1 2 3 4 5 6) => (1 2 3)
```

(prin [U:cualquiera]) : cualquiera Fsubr
El valor de U es impreso en la pantalla con los caracteres especiales precedidos por el caracter escape (!). Prin devuelve el valor de U. Prin puede tener cualquier numero de argumentos.

(princ [U:cualquiera) : cualquiera Fsubr
El valor de U es impreso en la pantalla sin caracteres escape. Princ devuelve el valor de U. Princ puede tener cualquier numero de argumentos.

(print [U:cualquiera]): cualquiera Fsubr
Es valor de U es impreso en la pantalla con los caracteres especiales precedidos por el caracter escape (!) y al final del mensaje se añade un caracter de salto de linea (LF). Print devuelve el valor de U. Print puede tener cualquier numero de argumentos.

(printc [U:cualquiera]) : cualquiera Fsubr
Es igual que print pero los caracteres especiales se escriben sin ir precedidos de escape. Printc puede tener cualquier numero de argumentos

---

(progn [U:cualquiera]) : cualquiera Fsubr
U es un conjunto de instrucciones que son ejecutadas de manera secuencial. El valor devuelto es el de la ultima expresion evaluada.

(put U:id IND:id PROP:cualquiera) : cualquiera Subr
El indicador IND con la propiedad PROP es puesto en la lista de propiedades de U. Si la accion de put se ejecuta el valor devuelto es el valor de PROP. Si U o IND no son identificadores se producira un error y no se instalara ninguna nueva propiedad. Put NO puede usarse para definir funciones. Ver tambien get y plist.

(quote U:cualquiera) : cualquiera Fsubr
Devuelve el valor de U sin evaluar. La abreviatura de quote es la comilla simple ('). Ambos, quote y la comilla son equivalentes.

(quotient U:numero V:numero) : numero Subr
Devuelve el cociente de la division entera de U entre V. En resto de esta division puede obtenerse mediante la funcion remainder. Quotient producira un error si se V es cero o si U o V no son numeros.

(randomize U:numero) : numero Expr
Randomize se usa para resetear el valor interno usado por rnd.

(rdf F:id) : nil Subr
Lee y ejecuta el codigo LISP contenido en el fichero especificado. Si se encuentra un (stop) en el fichero el control pasa de nuevo a rdf.

(rds F:fichero) : numero Subr
Rds selecciona el fichero especificado para lectura y devuelve el numero de fichero asociado, este fichero debe haber sido previamente abierto con open. Ver tambien wrs.

(read [U:numero]) : cualquiera Subr
Read lee una lista o expresion. En una entrada que LISP utiliza normalmente para recibir informacion interactiva por parte del usuario. U debe de ser el numero de fichero que identifique de donde queremos leer el mensaje. Si U no aparece o es igual a 0 este mensaje se leera del teclado. Ver tambien readline y getchar.

(readline [U:numero]) : id Subr
Readline lee caracteres del fichero especificado por U (su numero-de-fichero), si U no existe o es igual a 0 estos caracteres se leeran del teclado. Todos los caracteres de entrada de readline hasta el siguiente CR (ASCII numero 10) seran ensamblados en un unico identificador y este es el valor devuelto por readline. Puede ser util usar ordinal o explode para extraer caracteres de este identificador.

(recive DIR:numero) : numero Expr
Recive es en cierto modo una expresion analoga a PEEK_L pero menos potente que ella. El LISP del QL no esta preparado para soportar enteros de palabra larga (32 bits) porque toda su aritmetica la realiza solo con 28 bits. Recive devuelve el valor de la palabra larga en la DIReccion especificada pero esta palabra debe ser un numero positivo y no exceder de la capacidad aritmetica del LISP. Ver tambien send.

(reclaim) : numero Subr
Fuerza a LISP a realizar un grabage-collection. El usuario puede desear hacer esto para comprobar cuanto espacio de trabajo le queda libre aun. El valor devuelto por reclaim es este espacio. Ver messoff.

(remainder U:numero V:numero) : numero Subr
Devuelve el resto de la division entera de U entre V. Se produce un error si U o V no son numero o si V es cero. Ver tambien quotient.

(remprop U:cualquiera IND:cualquiera) : cualquiera Subr
Borra la propiedad con el indicador IND de la lista de propiedades de U. Devuelve la propiedad borrada o nil si no existia el indicador IND.

(rnd) : numero Expr
Llama: peek_w.
Devuelve un numero entero aleatorio.

(repeat N:numero CUERPO:[cualquiera]): cualquiera Fsubr
El CUERPO es evaluado N veces y la ultima evaluacion de CUERPO es devuelta. Por ejemplo,

```
(repeat 4 (turn 90) (draw 100))
```

(reset) Subr

Reset pone a cero los contadores de time y getime. Puede usarse para saber cuanto tarda LISP en ejecutar una funcion, por ejemplo:

```
     (progn
        (reset)
        <funcion>
        (list (time) (getime)))
```

(reverse U:lista) : lista Subr

Devuelve U con los elemento del nivel superior revertidos. Ej.

```
     (setq l '(a b c))
     (reverse l) => (c b a)
     l => (a b c)
```

(reversewoc U:lista) : lista Subr

Es la version destructiva de reverse. Por ejemplo,

```
     (setq l '(a b c))
     (reversewoc l) => (c b a)
     l => (c b a)
```

---

rpar Variable

El valor inicial de rpar es el parentesis derecho, ')'.

(rplaca U:par-con-punto V:cualquiera): par-con-punto Subr

Reemplaza el car de U con V. Si U es (a . b) (rplaca U V) devuelve (V . b) y U toma ese valor, rplaca es, por tanto, una funcion destructiva. Por ejemplo,

```
     (setq a (list nil))
     (rplaca a a)
```

producira una estructura asi: ((((((... Un lazo sin fin del que habra que salir con CTRL-ALT.

(rplacd U:par-con-punto V:cualquiera) : par-con-punto Subr

Reemplaza el cdr de U con V. Si U es (a . b) (rplacd U V) devuelve (a . V) y U toma ese valor, rplacd es, por tanto, una funcion destructiva. Ver tambien rplaca.

(save U:nombre-fichero) : nombre-fichero Subr

Esta funcion salva una copia de todo el espacio de trabajo de LISP usado en el momento de ser llamada. Su argumento es el nombre de fichero (incluido el dispositivo) donde se va a salvar esta copia. El entorno de trabajo salvado con save puede ser posteriormente recuperado mediante load. Save es completamente indiscriminado: graba todas las variables y sus valores, todas las propiedades y todas las funciones. Por ejemplo:

```
     (save 'mdv1_extend)
```

grabara en el microdrive 1 un fichero llamado extend, que puede quizas contener un toolkit con el que deseemos comenzar todas las sesiones de trabajo. Si un fichero producido por save esta presente en el dispositivo por defecto seleccionado, al arrancar el LISP este fichero se cargara automaticamente. Asegurese de tener una copia de seguridad del fichero image original del LISP de Metacomco antes de intentar esto

---

(scale N:numero) : numero Subr

Scale cambia el factor de escala de la pantalla. El valor por defecto de N es 1000.

(screen OP:numero U:numero V:numero) : Subr

Screen proporciona acceso directo a todos los TRAPs de entrada/salida del QDOS. Para una informacion detallada ver el QL Programacion Avanzada (TRAP#3). He aqui algunos de los TRAPs mas utiles.

```
     (screen 12 color anchura)     Resetea el borde.
     (screen 16 x y)               Posiciona  el cursor en las coordena-
```

```
                                das de caracteres x e y.
     (screen 17 x nil)          Posiciona el cursor en la columna x.
     (screen 18 nil nil)        Salta a la siguiente linea.
     (screen 19 nil nil)        Espacio atras.
     (screen 23 x y)            Posiciona  el cursor en las coordena-
                                das de pixels x e y.
     (screen 24 h nil)          Hace un scroll h unidades.
     (screen 27 d nil)          Hace un pan d unidades.
     (screen 32 nil nil)        Borra la ventana.
     (screen 33 nil nil)        Borra parte superior de la ventana.
     (screen 34 nil nil)        Borra parte inferior de la ventana.
     (screen 35 nil nil)        Borra la linea del cursor.
     (screen 39 col nil)        Pone el color del papel.
     (screen 40 col nil)        Pone el color del strip.
     (screen 41 col nil)        Pone el color de la tinta.
     (screen 42 flash nil)      Modo parpadeo (1=activo 0=inactivo).
     (screen 43 modo nil)       Modo subrayado (1=activo 0=inactivo)
     (screen 44 modo nil)       -1 = xor tinta y papel
                                 0 = escritura normal
                                 1 = strip transparente
     (screen 45 ancho alto)     Pone tamaño de los caracteres.
```

Ver tambien window.

(sed U:expresion) Subr

Sed es una subfuncion utilizada por el editor de estructuras de LISP. Sed define los comandos a los que respondera el editor. Estos comandos son de un solo caracter, obtenidos mediante getchar. Los mas importantes son a, d y b. a y d hacen entrar a sed en un proceso recursivo que va editando bien el car o el cdr de la expresion previa. b da los saltos atras y hace volver a sed a la expresion previa. Si se llega al principio, o al final, de una lista mediante car o cdr el uso de estos comandos provoca que sed imprima un asterisco y los ignore. El comando r premite reemplazar la expresion que actualmente este considerando sed. c y x insertan y borran items de listas. Sed imprime de manera legible la expresion corriente al final de cada llamada a un comando. Hay, por supuesto muchos comando adicionales que pueden figurar en la estructura del editor (comandos de busqueda y saltos de larga distancia, por ejemplo). El editor basico funciona originalmente en una version codificada para aumentar su velocidad. De cualquier manera este editor puede extenderse facilmente. Una version en LISP del mismo forma parte de las rutinas de demostracion.

```
     (defun sed (a (q))
        (loop
           (setq q (princ (getchar)))
           (until (eq q 'b) a)
           (SETQ a
           (cond
               ((eq q 'r) (terpri) (read))
               ((eq q cr) (superprint a) a)
               ((eq q 'c) (terpri) (cons (read) a))
               ((atom a) (princ '*) a)
               ((eq q 'd) (cons (car a) (sed (cdr a))))
               ((eq q 'a) (cons (car a)) (cdr a)))
               ((eq q 'x) (cdr a))
               (t (princ '?) a)))))
```

(send DIR:numero U:numero) : numero Expr

Llama: bin, dec, nthcdr, prelist.

Send es una funcion similar a POKE_L pero menos potente. El LISP del QL no soporta enteros de palabra larga (32 bits) ya que toda su aritmetica se realiza con 28 bits. Send pokea una palabra larga (U) en la -DIReccion especificada pero U debe de ser un entero positivo entre 0 y 536870909.

(set EXP:id VALOR:cualquiera) : cualquiera Subr

El efecto de set es reemplazar el item ligado al identificador por VALOR. Exp debe ser un identificador y no debe evaluar a t o nil (ya

que los valores de t y nil no pueden cambiarse), de nor ser asi se producira un error. Ver tambien setq.

(setdifference U:lista V:lista) : lista Expr
Llama: wdelete.
Setdifference realiza la diferencia de conjuntos. Entrega lo que queda de la lista U despues de eliminar de ella los elementos que aparecen tambien en V.

(setq VARIABLE:id VALOR:cualquiera) : cualquiera Fsubr
(setq <cualquier-cosa>) es equivalente a (set '<cualquier-cosa>), de hecho setq es la abreviatura de setquote. El valor de VARIABLE es reemplazado por VALOR. VARIABLE debe ser un identificador no igual a t o nil. Ver tambien set.

(smalerp U:atomo V:atomo) : booleana Expr
Llama: biggerp.
Smaller es una funcion que se usa para comparar cadenas alfanumericas. Si ambos, U y V, son numeros smallerp se comportara exactamente igual que lessp, sino procedera de por orden alfabetico de modo que una palabra que empieze por b sera menor que otra que empieze por a, si dos palabras coinciden en todas sus letras sera menor la que tenga un mayor numero de caracteres. Tambien es importante si las letras son mayusculas o minusculas son menores que las mayusculas. Por ejemplo

```
     (smallerp 'tortazo 'bofetada)  => t
     (smallerp 'abcd 'abc)          => t
     (smallerp 'puñetazo 'puñetazo) => nil
```

(sound U:numero V:numero) : no implementada Subr
Sound se utiliza para hacer sonar el zumbador.

(stop) Subr
Cuando stop es llamado desde el teclado de modo normal produce el abandono del LISP y el retorno al SuperBasic. Llamado desde un rdf stop devuelve el interprete al estado en que se encontraba justo antes de la llamada rdf. La salida al SuperBasic usando stop es irrevocable, una vez lo hayamos hecho perderemos todo nuestro trabajo que no haya sido grabado irreparablemente; por eso stop debe usarse con cuidado. Para salir temporalmente al SuperBasic sin perder nuestro trabajo podemos usar CTRL-C.

(sub1 U:numero) : numero Subr
Sub1 devuelve el valor de su argumento decrementado en 1. Ver add1.

(sublis U:a_lista V:lista) : lista Expr
Sublis es una funcion similar a subst pero permite hacer muchas sustituciones a la vez. U es una lista formada por pares con punto y V es la lista donde hay que hacer las sustituciones. Ejemplo:

```
     (sublis '((rosas . violetas) (rojas . azules))
               '(las rosas son rojas)) => (las violetas son azules)
```

(subset U:cualquiera V:lista) : lista Expr
Llama: consr, ncons.
Subset toma el predicado U y devuelve una lista formada por todos los elementos de V que satisfacen ese predicado. Ver tambien find-if.

(subrp U:cualquiera) : booleana Subr
Subrp testea si su argumento es o no el punto de entrada de una porcion de codigo maquina correspondiente a una funcion normal de LISP. De ser asi devuelve t, sino nil. Ver tambien fsubrp.

(subst U:cualquiera V:cualquiera W:cualquiera) : cualquiera Subr
El valor devuelto es es resultado de sustituir por U todos los V de de W.

(superprint U:cualquiera) : nil Subr
Imprime U con un formato de margenes con sangrado para facilitar su lectura e interpretacion. Las rutinas de demostracion que acompañan al interprete incluyen una version en LISP de esta funcion.

---

t Identificador Especial
El atomo "t" es la representacion standart en LISP para "verdadero" (en ingles true). T no debe ser usado como nombre para una variable.

(terpri) : nil Subr
Terpri provoca que el cursor salte al principio de la siguiente linea de caracteres.

(time) : numero Subr
Devuelve el tiempo transcurrido (excluido aquel empleado en el grabage collection) desde el ultimo (reset). Este tiempo viene dado el unidades de una centesima de segundo.

(times [U:numero]) : numero Fsubr
Devuelve el producto de todos sus argumentos.

(trace U:id) : id Expr
Activa el modo traza para la funcion U. Por ejemplo, si el usuario teclea

```
     (trace append)
```

y luego

```
     (append '(a b c) '(1 2 3))
```

el sistema respondera

```
     append = ((a b c) (1 2 3))
     append = (a b c 1 2 3)
```

donde la primera linea muestra la lista de los argumentos con los que se llama a append y la segunda linea muestra lo que devuelve. Ver tambien untrace. El intentar "rastrear" funciones usadas por trace o untrace o funciones que no evaluan sus argumentos puede causar problemas.

(turn U:numero) : numero Subr
Turn gira hacia la derecha la tortuga grafica U grados. Ej.

```
     (turn -90)  gira la tortuga 90º hacia la izquierda
     (turn 180)  invierte el sentido de orientacion de la tortuga
```

(turnto U:numero) : numero Subr
Turnto orienta la tortuga hacia el angulo especificado. Ej.

```
     (turnto 0) orienta la tortuga directamente hacia la parte superior
                de la pantalla.
```

---

undefined Identificador especial
Cuando se crea un nuevo identificador mediante read, getchar, character, readline o implode se le da el valor undefined (indefinido). Los atomos con el valor undefined o con su lista de propiedades vacia no aparecen en la lista de identificadores devuelta por oblist y pueden ser borrados por un grabage collection si ninguna estructura de datos hace referencia a ellos.

(until COND:cualquiera [V:accion]) : cualquiera Fsubr

Until se usa conjuntamente con loop. La condicion es evaluada y si resulta ser igual a nil until simplemente devuelve ese valor, nil. Si COND no es igual a nil ocurre lo siguiente:
i) Los valores pasados despues de la condicion son evaluados uno por uno y el ultimo es devuelto como valor de until. Si V no existe se devuelve el valor del predicado COND.
ii) Se pone un indicador que indica que loop debe terminar lo antes posible. Loop no terminara hasta que no este listo de nuevo para evaluar su primer argumento.

(untrace U:id) Expr
Desactiva el modo traza para la funcion U. Ver trace.

(usr U:numero) : no implementada Subr

(vdu U:numero [V:numero]): Fsubr
Todos los argumentos de vdu deben ser numeros. Esta funcion envia todos los numeros que se le pasen secuencialmente a la pantalla. Por tanto, (vdu 65 66 67) imprimira el mensaje ABC.

(wait U:numero) : t Expr
Wait es una funcion que detiene el funcionamiento de un programa durante U segundos.

(wdelete U:cualquiera V:lista) : lista Expr
Wdelete borra del nivel superior de V todos los elementos que sean iguales a U. Ejemplo:

```
(wdelete 'a '(1 2 a 3 4 a 5)) => (1 2 3 4 5)
```

(while COND:cualquiera [V:accion]) : cualquiera Fsubr
While se usa conjuntamente con loop. La condicion es evaluada y si no es igual a nil entonces el bucle continua. Si la condicion es igual a nil los valores de V son evaluados y el valor devuelto por while sera el valor del ultimo predicado evaluado. Ver until para mas informacion, while es la funcion complementaria de until.

(window CODIGO:numero ARGS:(l h x y) ANCHO:numero COLOR:numero) : Fsubr
CODIGO es un numero decimal. El hexadecimal equivalente puede encontrarse en el libro QL Programacion Avanzada de Adrian Dickens. La lista (l h x y) contiene cuatro numeros: 'l' es el largo, 'h' la altura de la ventana, 'x' e 'y' las coordenadas que decriben la nueva ventana. Los numeros de esta lista se usan para alterar una ventana desde el LISP. Los dos argumentos finales son opcionales y se usan para establecer el ancho y el color del borde. Por ejemplo;

```
(window 10 '(0 0 0 0))        devuelve  el tamaño actual de la ven-
                              tana  y  la posicion  del  cursor en
                              pixels.
(window 11 '(0 0 0 0))        idem  pero en coordenadas de caracte-
                              res.
(window 13 '(l h x y) w c)    redefine la ventana actual.
(window 30 '(0 0 0 0) 0 x)    mueve x pixels a la derecha la linea
                              del cursor.
(window 31 '(0 0 0 0) 0 x)    mueve x pixels a la derecha el final
                              de la linea del cursor.
(window 46 '(l h x y) nil c)  dibuja un bloque rectangular solido.
```

(write CANAL:cualquiera [ARGS:cualquiera]) : cualquiera Fsubr
Write es como print excepto que la salida no es a la pantalla sino al canal que indique el identificador de fichero, CANAL. Ver open y close. Write puede tener cualquier numero de argumentos.

(write0 CANAL:cualquiera [ARGS:cualquiera]) : cualquiera Fsubr
Write0 es como prin excepto que la salida no es a la pantalla sino al canal que indique el identificador de fichero, CANAL. Ver open y close. Write0 puede tener cualquier numero de argumentos.

(writec CANAL:cualquiera [ARGS:cualquiera]) : cualquiera Fsubr
Writec es como printc excepto que la salida no es a la pantalla sino al canal que indique el identificador de fichero, CANAL. Ver open y close. Writec puede tener cualquier numero de argumentos.

(writeo CANAL:cualquiera [ARGS:cualquiera]) : cualquiera Fsubr
Writeo es como princ excepto que la salida no es a la pantalla sino al canal que indique el identificador de fichero, CANAL. Ver open y close. Writeo puede tener cualquier numero de argumentos.

(wrs U:id-fichero) : cualquiera Subr
Wrs selecciona el fichero especificado para escritura y devuelve el identificador de fichero de U. El fichero U debe de estar abierto antes de realizar un wrs sobre el. Para abrir el fichero U se usa open que es la funcion que crea el identificador de fichero que servira de argumento a wrs. Ver open y rds.

(xtab U:numero) : nil Expr
Imprime U espacios al comienzo de una nueva linea.

(zerop U:numero) : booleana Subr
Devuelve t si U es cero, en caso contrario devuelve nil. Ver onep.

Paquete de desarrollo QL LISP Apendice A

El programa LISP_BOT.

Existen en el LISP del QL determinadas funciones que solo pueden implementarse en codigo maquina. Pero meterse a modificar el interprete con un desensamblador no solo seria un autentico "puro" sino una transgresion al copyright de Metacomco. Por ello la solucion adoptada en el L-TOOLKIT para implementar estas rutinas es la de crear un job auxiliar que lleve a cabo estas tareas, ese es el LISP_BOT. Esto no pasa de ser una chapuza ingeniosa, pero funciona.

El LISP_BOT esta escrito en SuperBASIC pero es necesario compilarlo para que funcione correctamente. El L-TOOLKIT incluye ambas versiones, en BASIC, LISP_BOT_bas, y en codigo maquina, LISP_BOT_exe.

El LISP_BOT_exe trabaja de la siguiente forma:

En primer lugar EXECuta el interprete LISP y reserva 100 bytes de memoria. La direccion de inicio de esta area reservada se POKEa en la direccion 131072 de la memoria (primera direccion de la pantalla). Esta area reservada de memoria sera la que utilizen el LISP_BOT y el interprete para comunicarse. Una vez inicializado el toolkit sera leido el valor de la direccion 131072 (palabra larga) y almacenado en la variable especial link, por eso NO SE DEBE ALTERAR LA PANTALLA ANTES DE INICIALIZAR EL TOOLKIT NI ALTERAR EL VALOR DE LA VARIABLE LINK porque eso podria conducir a un fallo total del sistema si empleamos la funcion call con posterioridad.

Una vez que tenemos ambos jobs, el LISP_BOT y el LISP, funcionando LISP_BOT chequea periodicamente los valores del area reservada y actua en consecuencia. El formato de este area reservada es el siguiente.

byte 0 : codigo de operacion.
byte 1 : sin usar.
bytes 2 y 3 (palabra): error devuelto.
bytes 4 al 51: datos pasados a LISP_BOT.
bytes 52 al 100: respuesta de LISP_BOT.

Los codigos de operacion validos en la version 1.0 del LISP_BOT son:

byte0=1 : Funcion CALL. La direccion de llamada se pasa en la palabra larga de direccion base_area+4.

byte0=2 : Funcion ALLOCATE. Es numero de bytes a reservar se pasa en la direccion base_area+4 (larga). La direccion efectiva donde comienza el area que hemos reservado se devuelve en base_area+52 (larga).

byte0=3 : Funcion DEALLOCATE. Su parametro es base_area+4= direccion efectiva a desasignar.

byte0=4 : Funcion KEYROW. La fila a leer se pasa en el byte base_area+4. El valor devuelto se coloca en la direccion base_area+52.

De este modo LISP_BOT se mantiene ejecutandose en baja prioridad hasta que valor del byte 0 del area reservada se vuelve distinto de 0. Entonces ejecuta la operacion pertinente, devulve los valores debidos y vuelve a poner en valor del byte0 a 0.

Es importante darse cuenta de que la comunicacion entre ambos jobs esta totalmente desprotegida, un fallo en los parametros de llamada produciria un error en el LISP_BOT y este se borraria, arrastrando con el al LISP, del que es propietario, y haciendonos perder todo nuestro valioso trabajo. Esto podria solucionarse utilizando la instruccion WHEN ERROR (ver QL Programacion Avanzada, Adrian Dickens, pags 361-362) en el LISP_BOT. La palabra que forman los bytes 2 y 3 del area reservada y la funcion errorgen del L-TOOLKIT estan especialmente pensadas para ello.

El LISP_BOT necesita para funcionar correctamente algunas extensiones, ya sean del TURBO TOOLKIT o del TOOLKIT II.

El programa LISP_LIB.

Este programa, a diferencia del anterior no necesita ser compila- do, aunque requiere algunas extensiones para el acceso aleatorio de ficheros. Estas son las extensiones POSITION y SET_POSITION del TURBO TOOLKIT o sus equivalentes BGET y BPUT del TOLLKIT II.

Lo primero que el programa nos pide al empezar es el nombre del fichero del cual vamos a extraer las rutinas es LISP. Estas rutinas deben comenzar con DEFUN y estar completamente bien escritas. El nombre de fichero debe terminar en _lsp. Si no acabara en estos 4 caracteres el LISP_LIB los añadiria automaticamente. Si tecleamos ENTER sin ningun nombre el programa se detendra.

Despues de eso hay que introducir el nombre de las rutinas que deseamos extraer de nuestro fichero, no importa si son escritas en mayusculas o minusculas pero es importante notar que si extraemos funciones del L-TOOLKIT que llamen al LISP_BOT tambien se debe incorporar al nuevo programa el PROGN que encabeza el L-TOOLKIT y que asigna el valor de la variable link. Cuando hayamos finalizado con las rutinas hay que teclear ENTER.

Despues de eso el programa nos pide el nombre del fichero donde van a ir a parar las rutinas extraidas. Este nombre tambien debe finalizar en _lsp.

Nuestra ultima opcion es volver al principio o borrar el programa.

L-TOOLKIT
Sergio Montoro Ten
Madrid, 10 de diciembre de 1989

COMENTARIOS DE PROGRAMAS

PROGRAMA : 3D PRECISION
DISTRIBUIDOR : DIGITAL PRECISION,
222 The Avenue,
London E4 9SE (UK)
PRECIO : 50 LIBRAS

Hace mucho tiempo que no he comentado un programa comercial. La respuesta es muy sencilla. Normalmente solamente suelo comentar aquellos programas que he comprado o tenia alguna curiosidad por ellos. Y durante los últimos años, mi colección de programas ha aumentado a un ritmo que me era totalmente imposible probarlos a fondo. Tal es el caso que en más de la mitad de los programas que he recibido en los últimos 15 meses, simplemente he probado si los ficheros estaban bien y han sido archivados en el almacén en espera que algún dia sean usados por alguien que los necesite (normalmente casi nunca soy yo). La cuestión es que hay tantos programas comerciales sin comentar que algunos no hace falta comentarlos porque ya no se comercializan, pero creo que habria de comentarlos por lo menos para quue alguien conozca su existencia.

En este comentario le ha tocado el turno al 3D PRECISION, el cual es todavía uno de los programas más actuales del mercado.

3D PRECISION es un programa que nos ayuda a diseñar y manipular objetos en 2D y 3D en un color o combinaciones de colores en el ordenador QL, THOR y compatibles.

Consiste en un 3D Editor, un Superbasic Toolkit y un assembler Toolkit.

Nuestros objetos se encuentran junto a una camera imaginaria la cual los observa, y son situados en un mundo en tres dimensiones con las coordenadas X, Y y Z en un rango de -32768 a 32767 en números enteros (en la práctica el rango es muchisimo menor).

Si solamente nos interesa el diseño de objetos en 2D o 3D, con usar el Editor es suficiente, que consiste en un programa multitarea dividido en varias ventanas: Dibujo, Ayuda, Comandos e Información.

Con el programa vienen 2 DEMOs: una en Assembler y otra en Superbasic. La primera es muy rápida y bastante simplona. La segunda es más compleja, y este es el listado en Superbasic:

```
30 GSTART 7,6
40 LOADOBJECTS
50 GWINDOW 256,128,256,0,128,320,17000:GMODE 8:MODE  8
52 OPEN#3,SCR_256X128A0X0:INK#3,7:CLS#3
54 OPEN#4,SCR_512X128A0X128:PAPER#4,0:INK#4,7:CLS#4
60 GCAMCEN 100,-225,-100
90 :
100 REPeat LOOP2
105 T=0:T1=1:T2=1
110 REPeat LOOP1
112   IF T>1 THEN AT#3,3,0:PRINT#3,T
114   SELect ON T
116   =0:LISTPRG:GSCALE 128,320
118   =1:INITCAR1:CLS#3:GCLS
120      PRINT#3," YOU ARE SITTING"
122      PRINT#3," IN A CAR."
124   =1 TO 313:MOVECAR1 5:MOVECAR2 6:MOVECAMERA 5:GMAKE:GCLEAR:GPLOT
126   =314:CLS#3:GCLS
128      PRINT#3," IT IS EASY TO CHANGE"
130      PRINT#3," THE VIEWPOINT."
140   =315 TO 470:GROTATE GOBJXA(6),GOBJYA(6),GOBJZA(6)
141    MOVECAR1 5:MOVECAR2 6:MOVECAMERA 6:GMAKE:GCLEAR:GPLOT
142   =471:CLS#3:GCLS
144    PRINT#3," TO ANY OTHER SET OF"\" CO-ORDINATES."
146    GSCALE 128,128:GPLACE -16200,4000,-11500:GROTATE -55,-55,55
148    GMAKE:GPLOT
150   =472 TO 570:MOVECAR1 5:MOVECAR2 6:GMAKE 5 TO 6:GCLEAR 5 TO 6:GPLOT 5 TO 6
180   END SELect
190   T=T+1:T1=T1+1:T2=T2+1
192   IF T=571 THEN PAUSE 500:EXIT LOOP1
194 END REPeat LOOP1
196 END REPeat LOOP2
197 CLOSE#3:CLOSE#3:GCLOSE
198 :
```

```
300 DEFine PROCedure MOVECAMERA(NO1%)
310   IF NO1%=5 THEN MOVECAR1 0:ELSE MOVECAR2 0:END IF
312   GPLACE GOBJX(NO1%)-100,GOBJY(NO1%)+225,GOBJZ(NO1%)+100
314 END DEFine
316 :
320 DEFine PROCedure INITCAR1
324   GCLS:T1=1
326   GPLACE -2800,225,-7200:GPLACE 5,2800,0,-7200:GPLACE 6,-16000,0,-13400
328   GROTATE 0,0,0:GROTATE 5,0,0,0:GROTATE 6,0,270,0
330 END DEFine
332 :
334 DEFine PROCedure INITCAR2
336   GCLS:T1=1
338   GPLACE -12000,16000,-11000:GPLACE 5,2730,0,-7200:GPLACE 6,-16000,0,-13400
340   GROTATE 0,270,0:GROTATE 5,0,0,0:GROTATE 6,0,270,0
342 END DEFine
344 :
350 DEFine PROCedure LISTPRG
351   MODE 4
352   PRINT#3," THIS PROGRAM "
354   PRINT#3," DEMONSTRATES"
356   PRINT#3," THE BASIC TOOLKIT."
358   PAUSE 200:LIST#4:PAPER#4,0:PAUSE 200:MODE 8
360 END DEFine
362 :
400 DEFine PROCedure MOVECAR1(NO1%)
410   SELect ON T1
415   =65 TO 85:GROTATE NO1%,0,13.5/(ABS(T1-71)+1.02)+1+GOBJYA(NO1%),0
420   =86:GROTATE NO1%,0,90,0
425   =216 TO 239:GROTATE NO1%,0,6/(ABS(T1-224)+1.02)+1+GOBJYA(NO1%),0
430   =240:GROTATE NO1%,0,148,0
435   =330 TO 348:GROTATE NO1%,0,-(8/(ABS(T1-338)+1.02)+1)+GOBJYA(NO1%),0
440   =349:GROTATE NO1%,0,90,0
445   =421 TO 445:GROTATE NO1%,0,8/(ABS(T1-429)+2)+2+GOBJYA(NO1%),0
450   =446:GROTATE NO1%,0,180,0
455   =510 TO 543:GROTATE NO1%,0,4/(ABS(T1-522)+2)+2+GOBJYA(NO1%),0
460   =544:GROTATE NO1%,0,270,0
465   =787 TO 800:GROTATE NO1%,0,4/(ABS(T1-791)+2.5)+1.2+GOBJYA(NO1%),0
470   =801:GROTATE NO1%,0,309,0
475   =843 TO 892:GROTATE NO1%,0,1+GOBJYA(NO1%),0
480   =893:GROTATE NO1%,0,0,0
482   =947:T1=1:GPLACE NO1%,2800,0,-7200
486   END SELect
488   GMOVE NO1%,50,0,0
490 END DEFine
495 :
500 DEFine PROCedure MOVECAR2(NO1%)
510   SELect ON T2
515   =16 TO 38:GROTATE NO1%,0,13.5/(ABS(T2-25)+1.02)+1+GOBJYA(NO1%),0
520   =39:GROTATE NO1%,0,0,0
522   =48 TO 55:GROTATE NO1%,0,-(15/(ABS(T2-51)+1)+5)+GOBJYA(NO1%),0
524   =56:GROTATE NO1%,0,270,0
530   =73 TO 92:GROTATE NO1%,0,3+GOBJYA(NO1%),0
532   =93:GROTATE NO1%,0,330,0
534   =115 TO 125:GROTATE NO1%,0,-5+GOBJYA(NO1%),0
540   =126:GROTATE NO1%,0,270,0
542   =157 TO 166:GROTATE NO1%,0,-9+GOBJYA(NO1%),0
544   =167:GROTATE NO1%,0,180,0
546   =200 TO 209:GROTATE NO1%,0,-5+GOBJYA(NO1%),0
548   =210:GROTATE NO1%,0,130,0
550   =222 TO 230:GROTATE NO1%,0,-4+GOBJYA(NO1%),0
560   =231:GROTATE NO1%,0,90,0
570   =300 TO 305:GROTATE NO1%,0,-13+GOBJYA(NO1%),0:GMOVE NO1%,-12,0,0
572   =306:GROTATE NO1%,0,0,0
576   =316:GPLACE NO1%,-11900,0,-10700:T2=41
578   END SELect
580   GMOVE NO1%,150,0,0
590 END DEFine
595 :
1100 DEFine PROCedure LOADOBJECTS
```

```
1110   GLOADOBJ "flp1_BASICDEMO1_OBJ3D"
1120   GLOADOBJ "flp1_BASICDEMO2_OBJ3D"
1130   GLOADOBJ "flp1_BASICDEMO3_OBJ3D"
1140   GLOADOBJ "flp1_BASICDEMO4_OBJ3D"
1150   GLOADOBJ "flp1_BASICDEMO5_OBJ3D"
1160   GLOADOBJ "flp1_BASICDEMO5_OBJ3D"
1170 END DEFine
```

Con este programa de demostración se consigue un efecto de movimiento primero desde el punto de vista del interior de un coche y finalmente desde el cielo. Y para haber sido escrito en Superbasic, me parece increiblemente fantástico como con esas instrucciones se puede crear algo tan complejo y bastante rápido.

En la cuestión de Toolkit, aqui hay para todos los gustos, programadores en assembler (o FORTH, C,...) y Superbasic, siendo la cantidad de comandos nuevos increiblemente grande, como siempre (esto último suele dar dolor de cabeza). Creo que se trata del mejor o único toolkit que existe para la creación de todo lo que se nos pueda pasar por la imaginación en tres dimensiones.

S. Merino, 6/12/1989.

BBS

LA NORMA CIITT V-24

Si en un intento de comunicación entre dos personas utilizamos los mismos sonidos,pero articulados de otra manera y con una sintaxis distinta,está claro que el entenderse es imposible.Lo mismo ocurre en el mundo de los ordenadores que,para un entendimiento entre ellos o entre ellos y sus perifericos no sólo es inprescindible que los hilos de conexión estén bien cableados sino que también las tensiones de las señales sean de un nivel correcto y que se respete su orden lógico en el tiempo además de que el protocolo(lenguaje logico) sea común para ambos.

Existe dos modos de comunicación:Modo paralelo y Modo serie.

Todas las comunicaciones se realizan en palabras de 8 bits independiente de que el ordenador sea de 8,16,32 o 64 bits.

En el modo paralelo existen,además de otros hilos para distintos controles entre máquinas,8 hilos en los que en cada instante está presente cada uno de los bits que forman la palabra.Este metodo es muy rápido (mas de 200.000 baudios) pero adolece que su radio de acción es muy corto (varias decenas de metros)por lo que queda exclusivamente reservado para comunicaciones locales.

En el modo serie los bits se transmiten de uno en uno,todos en fila india y en dos tipos de transmisión:Sincrono y asincrono.

En síncrono,el ordenador transmisor manda constantemente o a intervalos reducidos señal de su reloj para que el distante se sincronize constantemente con èl.Una vez sincronizados comienza el verdadero intercambio de información.

En asíncrono,al no existir impulsos de reloj , cada palabra de ocho bits se transmite independiente de las demás,formando por ella misma una transmisión completa.

Para ésto se le añade al principio un bit de arranque,que indica al receptor que va a comenzar un caracter . A continuación llegan los 8 bits significativos,llamados así por ser los que contienen la información.Y para terminar llega un impulso de parada que finaliza la comunicación.Hay variantes segun el modo escogido de trabajo en las que los bits significativos son 7 en vez de 8 y hay uno y medio o dos bits de parada.Tambien y como medio rudimentario de seguridad se puede añadir o no otro bit llamado de paridad.

En la norma V-24 se toma como base de partida la conocidisima norma EIA-RS 232 C que utiliza como conector el tipo "SUB-D" de 25 patas (pins) y con definición de tensiones entre -3 V c.c. a -25 V c.c. para el estado lógico "1"(llamado tambien MARCA) y +3 V c.c. a +25 V c.c. para el otro estado logico "0"(Conocido como ESPACIO).

De todas maneras las tensiones mas empleadas son +12 y -12 Voltios de corriente continua para las señales de ésta norma.

Debido a que la transmisión por dos hilos fisicos de una señal compuesta por

la conmutación mas o menos rítmica de los dos estados lógicos (0 y 1) que forman el contenido de la información o sea tensiones cambiantes entre +12 V y -12 V se ve rápidamente amortiguada por la impedancia caracteristica del medio de transmisión,éste modo de comunicación no alcanza mas de unos centenares de metros.

De ésta dificultad nació el MODEM (Modulador-Demodulador) que simplemente transforma los estados lógicos que recibe (+12V y -12V) en señales de frecuencia vocal.

El mes pasado vimos en la norma V21 como,por ejemplo, en modo LLAMADA el 0 lógico se transformaba en transmisión en 1.180 c/s.

Todas estas frecuencias usadas en las diferentes normas CCITT siempre están comprendidas en el paso de banda de un canal telefónico (300 c/s a 3.400 c/s) por lo que al estar diseñados los medios de transmisión telefónicos para dar la mejor respuesta/frecuencia y la mejor relación señal/ruido a ésta banda,es por lo que la comunicacion Ordenador-Modem-Medio de transmisión se realiza en mejores condiciones independiente que éste se realize por un cable,sistema de alta frecuencia,radio,fibra optica,etc. etc.

Para mi,la RS-232-C está claro que se diseñó principalmente pensando en manejar un modem ya que es curioso que la disposición del cable de unión ordenador-modem se realiza uniendo hilo a hilo (El 1 con el 1,2 con el 2,etc)todos y cada uno de los 25 hilos del conector.

Por supuesto al modem distante,en su recepción,sólo le llegan impulsos de frecuencia vocal formados por las diferentes frecuencias de marca y espacio que él transforma otra vez en tensiones continuas de +12v y -12v,con lo que se restituye la señal original como si se tratase de la de un ordenador muy cercano.

A efectos de señales electricas de +12V y -12V hemos hecho desaparecer de golpe toda la distancia fisica de la linea telefonica ya que el ordenador siempre sigue viendo en el modem el periferico que habla en el mismo idioma electrico que él y sólo ve el reflejo del ordenador distante como si realmente fuera muy cercano.

Ahora vamos a describir la función de los pricipales pins del conector con especificación de su uso y circuito CCITT asociado.

Pin 1.-Es la masa electrica de la alimentación (Cto 101)
Pin 2.-(TD)El ordenador manda datos para que el modem los transmita(Cto 103)
Pin 3.-(RD)El modem manda al ordenador los datos recibidos(Cto 104)
Pin 4.-(RTS)El ordenador indica al modem se prepare para emitir(Cto 105)
Pin 5.-(CTS)El modem contesta que está preparado(Cto 106)
Pin 6.-(DSR)Modem encendido(CTO 107)
Pin 7.-(SG)Masa electrica de la señal(Cto 102)
Pin 8.-(DCD)El modem detecta portadora del modem distante(Cto 109)
Pin 20.-(DTR)El ordenador ordena al modem se conecte a la linea telefonica si se trata de un modem de conexión directa (Cto 108,2)
PIN 22.-(RI)El modem avisa al ordenador que se está recibiendo una llamada entrante en modo respuesta automatica.(Cto 125)

Todas éstas señales están en reposo cuando se mide nivel lógico alto o sea -12 V c.c.(Marca).

El resto de los pins o no se usan o están reservados a funciones de control de diferentes circuitos(Ejemplo:Control del canal de retorno,etc) y no los describo por ser éstos de uso menos general.

Hasta otra ocasión.

Saludos,

Antonio Rodriguez
Aptdo 2107.
30000.-MURCIA

COMENTARIOS SOBRE HARDWARE

HARDWARE : PLOTTER SILVER REED

En primer lugar, no tengo tal cacharro, pero puedo describirlo. El Silver Reed era comercializado a finales de 1988 y principios de 1989 por la firma Inglesa Strong Computer Systems por un precio bastante bajo regalando el

programa Technikit o el QLCADette (ambos programas están diseñados para usar el Plotter), y no sé si ese aparato es o era comercializado en España por alguna firma.

Un amigo compró uno de esos Silver Reed con el programa QLCADdette (invierno'89), pero estaba interesado en hacer un screendump del QL en modo 4. Escribí un pequeño programa en Superbasic que hacia un scanner punto a punto en la pantalla del QL e imprimia con el Plotter cada punto a su color o similar. La rutina, si llegó a correr bien, era tan lenta que nunca me he enterado si habia funcionado.

El Silver Reed visto a primera vista parece una máquina de escribir electrónica, pero posee un interruptor con posición NORMAL o PRINTER, y en el lateral derecho se esconde debajo de una tapa un interface paralelo Centronics.

Se puede trabajar en dos modos: Texto y Gráfico. El paso de un modo a otro se hace con un simple código de 8 bits (En decimal el 17 para texto y el 18 para gráficos).

Los comandos gráficos son:

Comando	Formato	Función
DRAW	Dx,y -999<x,y>999	Dibuja una linea desde la situación actual de la pluma al punto (x,y)
RELATIVE DRAW	J .x, .y -999<.x,.y<999	Dibuja una linea desde la actual situación al punto situado a la distancia de (.x,.y) (he sustituido con un punto el simbolo delta)
MOVE	Mx,y -999<x,y<999	Mueve, con pluma arriba, desde la actual situación de la pluma al punto (x,y).
RELATIVE MOVE	R .x,.y -999<.x,.y<999	Mueve con la pluma arriba desde la presente situación de la pluma al punto situado a la distancia de (.x,.y).
HOME	H	Mueve al punto origen (0,0) con la pluma arriba.
INITIALIZE	I	Ajusta el punto de origen (0,0) a la actual situación de la pluma.
PRINT	Pc1 c2 c3..	Imprime caracteres. c=caracteres Este comando es cancelado por CR(Carrier Return).
COLOUR	Cn	Cambia el color al especificado por n.
SCALE	Sn n=0-3	Cambia el tamaño del carácter al tamaño especificado por n
ROTATE	Qn n=0-3	Cambia la carácter dirección del carácter por la especificada por n.

"n" en el comando COLOUR/SCALE/ROTATE

n	0	1	2	3
C	negro	rojo	azul	verde
S	S size	M size	L size	M x 3
Q	normal	tumbado derecho	al revés	tumbado izquierdo

El simbolo delta significa relativo a donde se encuentra situada la pluma siendo en sus coordenadas (0,0) en ese caso relativo.

Un ejemplo que dibuja un triangulo escrito en Superbasic:

```
100 OPEN#5,ser1
180 PRINT#5,CHR$(18):PRINT#5,  "I"
190 PRINT#5, "M400,-400" : PRINT#5,"I"
200 REMark Dibuja un Triangulo
230 PRINT#5, "C1" : PRINT#5, "D400,0"
240 PRINT#5, "C2" : PRINT#5, "D200,200"
250 PRINT#5, "C3" : PRINT#5, "D0,0"
260 PRINT#5, "M-75,-150" : PRINT#5, "C0" : PRINT#5, "S2" :
    PRINT#5, "P*****  sample - 1  *****"
270 PRINT#5, "H" : PRINT#5, "M0,-999"
280 CLOSE#5
```

Se recomienda en el manual que cuando retornamos de modo gráfico a modo texto, introduzcamos los códigos CR (10) + LF (13). En la tabla de códigos de caracteres se encuentran todas la vocales acentuadas en minusculas y mayusculas más la ñ y Ñ, y otras consonantes raras.

También, debo decir que consta de 4 plumas con sus cuatro colores. Pero no sé ni cómo se recargan (eso habria que preguntarselo a un usuario de Silver Reed). Y que el papel a diferencia de la impresoras matriciales, se supone que el rodillo a base de fricción lo hace subir y bajar para colocar la pluma en el lugar correcto, según coordenadas gráficas.

Bueno, supongo que he escrito suficiente para que os hagais una idea de qué es un Plotter.

S. Merino, 8/12/1989

Z88

THREADED INTERPRETIVE LANGUAGES Vs fig-FORTH Z80

Unos tres dias antes de Navidad recibi el libro Threaded Interpretive Languages con una nota que decia que el Z-80 fig-FORTH Assembly Source Listing me lo enviarian más adelante, porque no lo tenian en stock. A partir de ahora vamos a llamar al libro TIL (Threaded Interpretive Languages) nombre por el cual se conoce a lenguajes como el FORTH, IPS y STOICS (los dos últimos son variantes del FORTH), los cuales están a caballo entre intérpretes y compiladores, pero son más rápidos que los compiladores y muchisimos más faciles de desarrolar.

El libro TIL ha sido escrito en 1978 cuando todavía no existia FIG, y el FORTH solamente habia hecho asomar el hocico.

He encontrado la lectura del libro muy fácil e interesante a pesar de estar escrito en Inglés. Solamente tengo que decir que antes de Nochebuena ya habia leido el libro entero y antes de fin de año ya tenia escrito mi Z88 FORTH v1.0 con Assembler Z80 en FORTH, el cual no he podido aún probar debido a que estoy esperando un ensamblador Z80.

Tengo intención de explicar como se escribe un TIL, pero no sé si me voy a lanzar ya a escribir mi FORTH de 32 bits no standard (el MERINO-FORTH) para mi QL o explicar las rutinas que usa el Z88 FORTH (el cual solamente ocupa menos de 4 Kbytes con un diccionario de más de 150 primitivas y secundarias + el assembler).

Para que hagais una idea de lo simple que es un TIL (un FORTH), todo el código del Inner Intérprete (el corazón que hace funcionar todo el sistema) solamente ocupa 36 bytes (COLON, SEMI, NEXT, RUN y EXECUTE). Lo demás es todo diccionario, variables del sistema y algunas llamadas I/O al sistema operativo OZ.

Lo peor vino cuando recibí, el 4 de enero, el Z-80 fig-FORTH Assembly Source Listing. Si hubiese pedido solamente eso, quizás hubiese tirado el proyecto al cubo de la basura. Se trata de un listado Assembler, casi sin comentar, de cerca de 100 páginas tamaño folio, el cual viene preparado para una versión del CP/M con discos de 8". Vamos, que hay tantas etiquetas, encima usa el Cromenco CDOS Z80 Assembler versión 02.15, ocupa 10 kbytes, psst,.... mejor olvidar al fig-FORTH, el cual es una joya prehistorica, pero que quizás el emulador SUCCESS del QL se lo pueda tragar. Claro que copiar cerca de 100 páginas de código assembler 8080/Z80 y sin garantias, es algo que hay que pensarselo muy bien (mi tiempo es oro). Lo bueno que tiene esta historia es que el objetivo de escribir mi Z88 FORTH se ha cumplido y he aprendido lo suficiente para escribir mi nueva versión FORTH para mi QL.

Hasta que no tenga ensamblado el Z88 FORTH y haga algunas pruebas para estar seguro que corre medianamente aceptable, no voy a entregar el código fuente (para curiosos), el programa BBC BASIC cargador, el código objeto del Z88 FORTH y el fichero ASM_FTH (el cual contiene el assembler Z80). Y naturalmente un manual que me va a costar sudor y sangre traducirlo del Inglés (el Z88 FORTH utiliza muchas utilidades del OZ)

El proyecto nuevo FORTH para el QL no tiene prisa, y voy a pedir la documentación actual del ANS FORTH STANDARD. Por si me interesa desarrollarlo en versión QL.

S. Merino, 5/1/1990